IBM

Reference Manual STRAP-II

7030 Assembly Program

IBM

Reference Manual STRAP-II

7030 Assembly Program

Contents

Introduction

An assembly program for an electronic computer is actually a translator; it translates a program from a language convenient for the programmer to use into a language that is easy for the computer to use. An assembly program, therefore, simplifies the writing of programs for a computer.

STRAP-II, the STRETCH Assembly Program, defines a language for programmers for the IBM 7030. The STRAP-II language is a symbolic language—it provides a complete set of mnemonics for the expression of machine instructions and it permits the use of symbolic names for the locations of data and instructions. STRAP-II also provides mnemonics for a large set of pseudo operations, defined by STRAP-II to simplify the definition of data and the issuing of directions to the assembly program itself.

STRAP-II is a large program; it must be run on a 7030 computer with a minimum storage capacity of 24,576 words. If the 7030 being used has a disk unit attached, STRAP-II will require three tape drives—one input tape and two output tapes. If there is no disk attached, seven tape drives must be available for assembly—one drive for the system tape, one for input, three intermediate tapes and two output tapes. In either case, the input tape, the output tape for peripheral punching and the output tape for peripheral printing of the listing are Spool Tapes that are part of the 7030 Master Control Program.

Any program that can be assembled by STRAP-I, an early assembly program for the 7030 computer that assembled programs on the IBM 704 that would be run at a later time on the 7030, can be assembled by STRAP-II without modification. Thus, all the specifications defined in the 704-709-7090 Programming Package Manual (IBM Form No. C22-6531-1) for STRAP-I are also applicable to STRAP-II, with the exception of the behavior of two pseudo operations that control the format of the output listing, and one new restriction of the use of radix 16.

STRAP-II is a more elaborate programming system than STRAP-I; its specifications contain several new features not previously available in STRAP-I. These new features remove some of the restrictions of STRAP-I, offer more flexibility, and in some cases provide for completely new functions to be performed.

The remainder of this manual is divided into two major sections. The first section will describe input format, the expression of machine instructions and pseudo operations, and the output format. The second section extends the description of some of the features in the first section to explain certain less frequently used but more advanced and complex procedures that are available to the programmer.

Section I

Input Format

Figure 1 illustrates the STRAP coding sheet (IBM Form No. X22-6798-0) with some STRAP statements written on it.

1	2 NAME 9	10
	ANYNAME	L(BU, 64, 8), DATA($X3), 7
	ADDER	+(BU, 8, 8), SINE8

FIGURE 1. STRAP Coding Sheet.

The STRAP coding sheet was designed to simplify the writing of instructions in a neat and orderly fashion. The coding sheet is divided into four fields:

1. Class (1 column)—used by STRAP to identify Master Control Program cards, continuation cards and comment cards.
2. Name (8 columns)—identifies the statement.
3. Statement (63 columns)—used to express a 7030 instruction or pseudo instruction.
4. Identification (8 columns)—identifies the card.

Card identification (columns 73-80) is reproduced on the output listing, but does not contribute any information to the assembly program for translating instructions.

The format of the coding sheet is directly related to the format of the symbolic input card (IBM card Form No. A36259). Both are divided into the same four fields. The coding sheet is most useful as a document from which the keypuncher can punch the program directly on to the input cards. The first instruction from the above illustration of the coding sheet is shown in Figure 2 as it would be punched on a STRAP input card.

One line on the coding sheet represents one punched card. Normally, one machine instruction or pseudo operation is written per line. A comment may follow any instruction. The beginning of a comment is signaled by the character ' (an 8-4 double punch); it is usually terminated by the end of the card.

```
CONV3 L(BU,64,8),TABLE'BEGIN CONVERSION
```

Comments are reproduced on the output listing, but do not affect the assembly program in any way. If an ' appears in the name field, the entire card block is treated as a comment—it is reproduced on the listing but it is not assembled.

Several statements may be written in the statement field of a single symbolic input card. Multiple statements are separated by the special character ; (an 11-0 double punch), which implies the end of a statement. Therefore, the character ; can never be used in a comment, except in a comment card block.

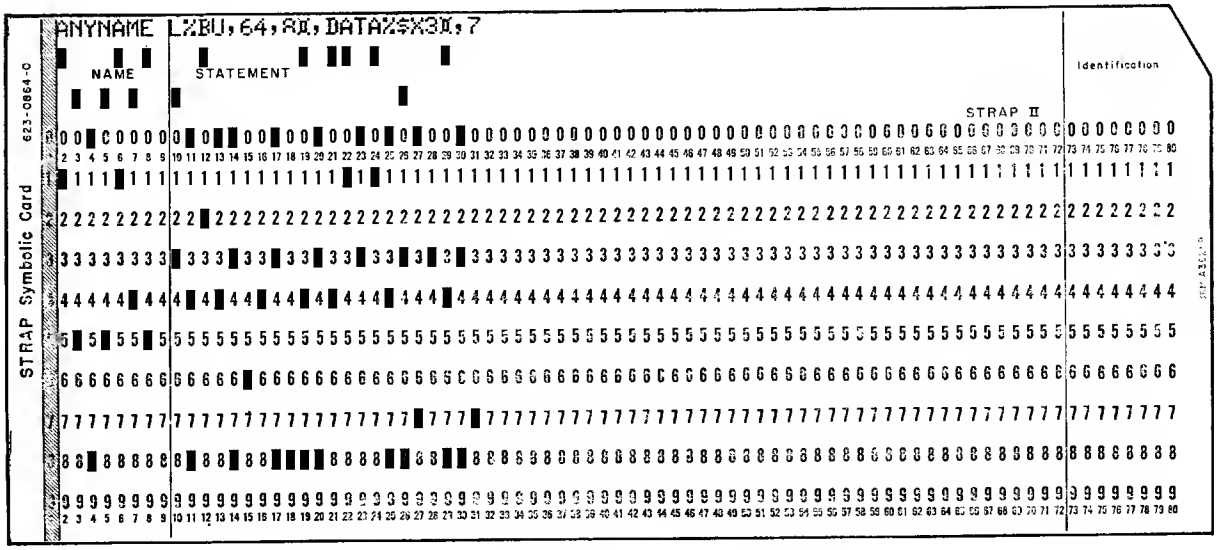


FIGURE 2. STRAP Symbolic Card.

BEGIN L(N), DATA; +(N), FIRST1; −(N), ANGLE

The number of instructions that may be written on one line is limited only by the number of columns available in the statement field of the card. The symbol in the name field of a card having more than one instruction in the statement field is associated with the first instruction only. The remaining instructions are treated as if they appeared on separate cards having blank name fields.

The name field and/or the statement field of a symbolic input card can be continued on subsequent cards by use of a continuation card. A continuation card is identified by an * punched in column one. In all other respects it is identical to the symbolic input card. In STRAP-II, a card block is defined as the initial symbolic input card plus all its continuation cards.

REALLONG L(N), DATAWORD; *(N), FACTOR
*NAME

If continuation cards are used to extend the name field, one restriction applies—the first character of the name must appear in the name field of the first card in the card block. A name, regardless of its length, is always attached to the first statement of the card block.

Expression of Machine Instructions

Symbolic machine instructions are written in the statement field of the coding sheet. Symbolic instructions are divided into several fields (operation mnemonic, data description, address, offset, etc.) by commas. These major fields may in turn be further divided into subfields or modified by expressions contained in parentheses, such as index register specifications, secondary operations in progressive indexing, and so on.

The format of the symbolic instruction varies with the class of STRETCH instructions to which it belongs. There are twelve symbolic instruction formats for STRAP-II.

1. Floating Point

   OP(dds), A_{18}(I)

   Example:

   ST(U), BUCKET($2)

   This instruction says, "Store the contents of the accumulator as an unnormalized floating point number in the storage location symbolized by BUCKET modified by index register 2."

2. Miscellaneous, unconditional branch, SIC

   OP, A_{19}(I)

   Example:

   B, START($X12)

   This STRAP instruction means, "Branch to, or transfer control to, the instruction whose location is symbolized by START modified by index register 12."

3. Direct Index Arithmetic

   OP, J, A_{19}(I) *or* OP, J, A_{18}(I)

   Example:

   LX, $3, XWORD($6)

   This instruction, when executed, tells the 7030 computer to, "Load index register 3 with the contents of the word found at the location symbolized by XWORD modified by index register 6."

4. Immediate Index Arithmetic

   OP, J, A_{19} *or* OP, J, A_{18}

   Example:

   V+I, $10, 1024

   The meaning of this instruction is "Add the address of this instruction to the value field of index register 10."

5. Count and Branch

   OP, J, B_{19}(K)

   Example:

   CB, $8, BEGIN($1)

   This instruction directs the computer to "Subtract one from the count field of index register 8, then test the count field. If it is not zero, branch to the location specified by the symbolic location BEGIN modified by index register 1. If the count field is zero, do not branch but proceed to the next instruction in sequence."

6. Indicator Branch

   OP, B_{19}(K)

   Example:

   BZM, ERROR($7)

   This instruction, whose operation code mnemonic is partially constructed from the name of the indicator, says "Branch to the instruction located at the location symbolized by ERROR modified by index register 7 if the Zero Multiply indicator is on. If it is not on, proceed to the next instruction in sequence."

7. VFL Arithmetic, Connect, Convert

   OP(dds), A_{24}(I), OF_{7}(I')

   Example:

   M+(BU, 24, 8), DUMMY($9), 6($4)

This variable field length operation says "Take the 24-bit unsigned field composed of 8-bit bytes found offset from the right end of the accumulator by an amount equal to 6 bits modified by index register 4 and add it to the field of the same length that is found beginning at location DUMMY modified by index register 9 in storage."

8. Progressive Indexing

$OP_1(OP_2)(dds)$, $A_{24}(I)$, $OF_7(I')$

Example:

ST(V+I)(BU, 24, 8), .30($8), 2($14)

This VFL instruction with progressive indexing illustrates the power of STRETCH instructions. The operation reads "Store the unsigned 24-bit field composed of 8-bit bytes that is found offset from the right end of the accumulator by 2 modified by index register 14 bits in the storage location specified by the value field of index register 8. Then increment the value field of index register 8 by 30 bits and proceed to the next instruction in sequence."

9. Swap, Transmit full words.

OP, J, $A_{18}(I)$, $A'_{18}(I')$

Example:

T, $2, TABLE1($3), TABLE2($4)

This transmit instruction is written to "Transmit the number of full words specified by the count field of index register 2 from the storage area beginning at location TABLE1 modified by index register 3 to the storage area beginning at TABLE2 modified by index register 4."

10. Branch on Bit

OP, $A_{24}(I)$, $B_{19}(K)$

Example:

BB, ONEBIT($5), FIXUP($9)

This instruction is interpreted to mean, "If the bit in storage whose location is ONEBIT modified by index register 5 is on, branch to the instruction at location FIXUP modified by index register 9. If this bit is not on, proceed to the next instruction in sequence."

11. Input-Output

$OP_1(OP_2)$, $A_{19}(I)$, $A_{18}(I')$

Example:

RD(SEOP), CHANX($6), CONWORD($9)

The meaning of this I/O instruction is "Read the unit connected to the channel symbolized by CHANX modified by index register 6 (or the last unit located on this channel if more than one unit is attached) according to the instructions contained in the control word addressed by CONWORD modified by index register 9."

12. Load Value With Sum

LVS, J, X_1, X_2, X_3, ...

Example:

LVS, $3, $5, $6, $7, $8

This instruction reads, "Add together the value fields of index registers 5, 6, 7, and 8 and store the sum in the value field of index register 3."

Each of these formats is a slight variation or expansion of the basic STRAP instruction format which is:

OP, A

The inclusion of index modification of the principal address expands this basic pattern to a STRAP format

OP, A(I)

used in Unconditional Branches, Indicator Branches and Miscellaneous instructions. Through the addition of the data description field

OP (dds), A(I)

we arrive at the format for floating point instructions. Adding to this format the offset specification and its index modifier

OP(dds), A(I), OF(I')

we develop the Variable Field Length format.

Other changes in the basic format yield the other STRAP formats. For example, the insertion of the J field to specify the index register that is being operated upon

OP, J, A(I)

becomes the basis for the Index Arithmetic and Count and Branch formats. Swap-Transmit, Input-Output, Bit Branching and the Load Value With Sum formats evolve from the basic STRAP format in similar fashion.

The major fields in any STRAP format are separated by commas. All of the fields shown for a particular format need not be written in every instruction. It is obvious, for example, that an offset would not always be specified in every VFL arithmetic statement. Therefore, a right-to-left drop-out order for major fields has been established; that is to say, missing fields are compiled by STRAP-II as if they contained zeros and were added at the end of the statement. A missing field that is always compiled in some standard fashion (in this case zero) is referred to as a null field. Our previous example of a VFL arithmetic instruction, now written with the offset field null, would appear

M+(BU, 24, 8), DUMMY($9)

and the instruction will be compiled by STRAP-II with zero offset.

The complete right-to-left drop-out of fields in a VFL statement is illustrated below to show the programmer how the expression of a STRAP statement may vary within the framework of the format for that class of instructions.

OP(dds), A(I), OF(I')
OP(dds), A(I), OF
OP(dds), A(I)
OP(dds), A

Notice that when even a complex format such as the VFL format is written to include only the essential information, the result is a statement that differs very little from the basic STRAP instruction format previously illustrated. It will be seen later that in actual practice, the (dds) field can almost always be eliminated in the instruction proper (see Data Description discussion).

A major field may be null even if other non-null fields follow. Such is the case if nothing but the comma denoting the field termination is written. Thus, a VFL instruction written with its address and index modifier null but with an offset specification following would appear as in the illustration below

OP(dds), , OF_7(I')

Note that it is only the presence of the comma that indicates the missing address field. If the comma were omitted, STRAP would assume that the offset field were null and would actually compile the offset specification as the address expression.

Some of the components of a major field can be made null simply by omission. We have seen, for example, that the offset specification in a VFL statement need not be indexed and can be written

OP(dds), A(I), OF

Similarly, the address expression need not be indexed, and can be written

OP(dds), A, OF

Obviously, if all the components of a major field are omitted (both offset expression and its index modifier, for example) the field is made null. This will normally be just what the programmer desires, but care must be taken if the null field occurs in the middle of the statement. As explained above, if the comma denoting the termination of the null field is also missing, the null field is assumed to be missing from the right hand end of the statement.

On the following pages, a detailed discussion of some of the major fields in the STRAP instruction formats is found. The three fields covered are the operation field, the address field and the offset field.

These fields are singled out because they are common to most instructions or because they illustrate important programming features or facilities of STRAP. The operation field is, of course, common to all instructions. The data description appears as a sub-field of the operation field in those instructions where it is appropriate. The address field is also common to all 7030 instructions, although it varies considerably in length. The address field typifies a field in which a wide variety of entries can be made. Rules for interpreting these entries and translating them for internal use are illustrated for address fields and hold true for most other instruction fields. The offset field is found only in variable field length instructions, but it is interpreted as an address field of an unusual length. It illustrates some unique index modification as well.

Two general comments are appropriate here. First, all 7030 instruction fields are unsigned. Any numeric entries that are negative are converted by STRAP and expressed as the two's complement of the entry. Second, all numeric entries in the illustrations are assumed to be written in the decimal radix. Entries in other radices are permitted in STRAP II if the radix is specified in a standard fashion (see Entry Mode in Section II of this manual).

Operation Field

STRAP-II provides a complete set of mnemonics for the expression of all 7030 instructions. The use of mnemonics is desirable from a programming standpoint because they make instructions brief to express, easy to remember and easy to recognize.

A complete list of STRAP-II mnemonics is given in Appendix A. A few rules may be noted in choice of mnemonics. First, the mnemonic is as brief as it can be and still unambiguously identify the instruction. Second, standard symbols are used for arithmetic operations— + for ADD, − for SUBTRACT, * for MULTIPLY and / for DIVIDE. Third, the receiving register (that is, the register that receives the result of the operation) in arithmetic operations is indicated by the letter to the left of the arithmetic symbol. In cases where the result is in the accumulator, the accumulator is assumed and is not mentioned in the mnemonic. For example, + is the mnemonic for straight ADD where the result is left in the accumulator, M+ is the mnemonic for ADD TO MEMORY and V+ means ADD TO VALUE. Fourth, certain basic operations may be altered to invoke immediate addressing by adding the suffix I as in V+I, ADD IMMEDIATE TO VALUE.

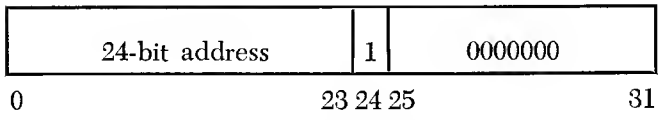
Mnemonics for STRAP-II pseudo operations may also be written in the operation field of a STRAP statement.

The formats for the expression of pseudo operations are similar in style to the STRAP instruction formats. (See Pseudo Operations.) A complete list of STRAP-II pseudo operation mnemonics is given in Appendix B.

A null operation code field occurs if the first character in a 7030 statement is a comma, as in this example:

```
, EXIT
```

STRAP treats a null operation field as a special case; it compiles the statement as a half word with a 24-bit address field, the 25th bit set to one and all the rest of the bits set to zero, thus:



This half word appears, when compiled, to be the first half of a full word instruction because of the one bit in bit 24 and the zeros following. This can be helpful to the programmer in some situations. For example, if it is desired to load the quantity compiled in the address field into the value field of an index register, LVE (the Load Value Effective instruction), which is indexable, can be used. This instruction examines the half word to determine the class of instructions to which it belongs. Since the half word resembles the first half of a full word instruction, LVE loads all 24 bits of the address field. If LV (Load Value) is used, 25 bits will be loaded (24 bits plus sign) and the one in the 25th bit makes the value appear negative. Therefore, caution must be used when creating value fields in storage by means of statements with null operation fields. An alternative method is available through use of the pseudo operation VF (see Pseudo Operations).

A secondary operation mnemonic may appear as a subfield of the operation field in progressive indexing. Here the secondary operation is enclosed in parentheses and follows the primary operation mnemonic.

Data Description (dds)

A second sub-field that may appear in the operation field of certain instruction formats is the data description. It is symbolized in the instruction formats above by the letters dds enclosed in parentheses. This field contains three specifications:

M	Use Mode
FL	Field Length
BS	Byte Size

These three specifications appear in the above order within parentheses and are separated by commas, thus:

```
(M, FL, BS)
```

The dds immediately follows the operation mnemonic, except in progressive indexing, where it follows the secondary operation.

A data description is required only by the floating point and VFL formats. In floating point instructions, the data description tells whether the instruction calls for normalized or unnormalized operations. Field length and byte size are not appropriate. In VFL statements, the data description specifies signed or unsigned binary or decimal operations. In addition, it describes the field length and byte size of the data to be operated upon. One additional mode, the properties mode, may appear in either type instruction as explained below. Here again, field length and byte size are not appropriate with the P mode.

STRAP-II provides seven brief mnemonics to designate a use mode. These are:

N	Normalized Floating Point
U	Unnormalized Floating Point
B	Binary (Signed)
BU	Binary Unsigned
D	Decimal (Signed)
DU	Decimal Unsigned
P	Properties Mode

The field length and byte size specifications are normally numeric entries, but they may be symbolized by the programmer, provided, of course, that the symbols are correctly defined elsewhere in the program. A typical floating point instruction with data description is illustrated below:

```
L (N), SINEX
```

The data description (N) indicates a normalized floating point data word located at SINEX is to be operated upon. In the following VFL instruction

```
L (BU, 30, 6), ADJUST
```

the data description describes the data at symbolic location ADJUST as binary unsigned, 30 bits in length, composed of 6-bit bytes. Note that in cases where the operation mnemonic is the same for VFL and floating point instructions, it is the data description that tells STRAP to which class the operation belongs and, hence, which operation code to compile.

A data description given with any of the four data entry or data reservation pseudo operations (DD-Data Definition, DDI-Data Definition Immediate, DR-Data Reservation and SYN-Synonym are all discussed in the Pseudo Operations section) is attached to the symbol in the name field of that statement, and is automatically invoked whenever that symbol appears in the principal address field of a 7030 instruction. Since

this is the usual practice, in straightforward programming it is unnecessary to write a data description in machine operations. When several symbols are joined arithmetically in an address field, the data properties of the last one written down are invoked for the statement.

When the data description *is* written as a sub-field in the operation field of a machine instruction, it *overrules* any other data description derived from a symbol in the address field *for that statement and that statement only.*

The mnemonic "P" in the mode field of a data description has this special meaning:

(P, NUT)

specifies that the data description associated with the symbol NUT is to be invoked as if it had been written out explicitly in this instruction. Thus, in an instruction, the properties mode invokes a data description that overrules any data description implied by a symbol in the principal address field.

Within a data description field, the usual right-to-left drop-out order holds (except that the mode field can never be null), so that a data description may appear in any of the following four forms:

(M)	Field length and byte size are null
(M, FL)	Byte size is null
(M, , BS)	Field length is null
(M, FL, BS)	

If the field length is null, a field length of zero (effectively 64 — see 7030 Reference Manual — except in the case of VFL immediate where it is 24) is compiled. However if the byte size is null, the byte size compiled by STRAP is a function of the mode specified.

MODE	STANDARD BYTE SIZE
D or DU	4
B	1
BU	8
N or U	Fixed format of 64 bits; field length and byte size not appropriate.

Cases can arise from programmer errors in which a data description and an operation are not mutually consistent (when the operation mnemonic specifies a floating point operation and the use mode in the data description is binary unsigned, for example). In this case the operation overrules.

Other cases can arise where there is no way for STRAP to obtain a data description from either the symbolic address or an explicit data description field. Three distinct situations can be encountered here:

1. The operation symbol can stand for either VFL or floating point operations (+, —, *, /). The operation is compiled as a VFL operation with the data description (BU, 64, 8).
2. The operation mnemonic can stand for a VFL operation only (M + 1, for example). The statement is assigned a data description (BU, 64, 8). If the operation is clearly VFL immediate, (BU, 24, 8) is assigned.
3. The operation mnemonic can only be a floating point operation (—A, or *NA). The operation is assembled as normalized floating point, except for the case of E + I (Add Immediate to Exponent) and its modified forms, where unnormalized is assumed.

If STRAP encounters any of the four irregularities described above, the indicated action is taken and an error message is printed on the output listing.

Address Field

The maximum core storage capacity of the IBM 7030 computer is 262,144 words (each word 64 bits in length) or 2^{18} distinct locations. Hence, 18 binary bits can unambiguously specify any word in 7030 memory.

Any single bit in core storage can occupy one of 64 positions within a word. Since it is a STRETCH convention to number bit positions in a word from 0 (the leftmost bit position) to 63 (the rightmost bit position), 6 binary bits are sufficient to specify any bit position within a 7030 word.

Clearly then, 18 + 6 = 24 binary bits are adequate to address a single bit anywhere in STRETCH core storage — the first 18 bits specify a full word and the last 6 bits specify a bit position within that word. Such a 24-bit binary address, when appearing in the address field of a 7030 statement, is known as a *standard binary bit address*, commonly abbreviated to "bit address."

In 7030 programming, the address of an instruction need specify only the leading bit of the operand since the field length of the operand is always known. If the operand is an instruction, a field length of either one-half word or a full word can be determined from the operation code. If the operand is data, the data description gives the field length; floating point data always occupies a full word, while the field length of VFL information is specified explicitly in the dds.

Certain rules for the location of data or instructions further simplify the addressing of operands. Thus, to address an index word, which is constrained to begin at a full word, only 18 bits are required. A 19-bit standard binary bit address is adequate to address any instruction, since instructions can only be located to

begin at half or full words. Other examples are seen below:

A VFL operand may begin anywhere in core storage — a 24-bit standard binary bit address is required.

A floating point operand must begin at a full word — this location can be specified in 18 bits.

I/O control words must begin at a full word — the location can be specified by 18 bits.

Index arithmetic operands can begin at either half or full word locations — 19 bits are sufficient to address any of these locations.

A floating point instruction needs only to address floating point data; hence the size of the address field of a floating point instruction is limited to 18 bits. A VFL instruction must be capable of addressing any field, in fact any bit, in storage. Its format, therefore, provides for a 24-bit address field. In general, the STRAP instruction formats are designed to provide the largest address field demanded by the operations of a particular class. When an instruction does not require a 24-bit address field, a smaller one is provided and the bits not used as part of the address field can be efficiently used in other fields of that instruction, leading to the variations in format we have already viewed.

It would be most difficult for the programmer to write 24-bit, or even 19-bit or 18-bit, binary addresses in his program. In place of a binary address, STRAP permits the programmer a choice of entries in address fields. He may write a

1. STRAP bit address
2. Integer
3. Programmer symbol
4. System symbol

Address fields are unsigned fields. When a negative quantity is expressed in an unsigned field, the two's complement of the quantity is computed and compiled by STRAP.

STRAP BIT ADDRESSES

A STRAP bit address provides a simple means of writing a standard binary bit address. Using this STRAP shorthand, the programmer writes two integers separated by a period, thus:

124.32

The integer to the left of the period specifies the word address portion, while the integer to the right of the period specifies the bit position within that word. If the example above appeared in the address field of a VFL instruction, STRAP would interpret it as "location 124, bit position 32 — the first bit of the second half word". *Note that the period is definitely not a decimal point.* This can be proven by the following illustration.

999.1= 999.01

A STRAP bit address is translated by STRAP and compiled as a 24-bit binary integer. The period that separates the two integers may be imagined to always line up between bit positions 17 and 18. If the address field of the instruction is 24 bits long, the binary integer is simply placed in that field. If the address field is smaller than 24 bits, the 24-bit standard binary address must be truncated before it is inserted. For a 19-bit address field, the rightmost 5 bits are dropped; for an 18-bit address field, the rightmost 6 bits are dropped. Our sample STRAP bit address, 124.32, would yield the proper meaning when inserted in a 19 or a 24-bit address field, but would be truncated to 124.0 for an 18-bit field.

The only restriction on the size of a STRAP bit address is that it must be able to be expressed in 24 binary bits. If a STRAP bit address is symbolized by A. B, then

$$64A+B < 2^{24}$$

The following three examples are all legal STRAP bit address representations of the same address.

505. 17= 500.337= 0.32337

INTEGER ADDRESSES

An integer, written without a period, may also be used to specify an address. STRAP also translates an integer into a standard binary bit address. However, the bit address equivalent is dependent upon the environment in which the integer is found. The operation determines the environment, that is, it determines the length of the address field. The integer specified is treated as an integer for that address field, i.e., the integer is converted to binary and inserted in the address field with the unit bit placed in the rightmost bit position of the *field.*

An integer can be interpreted by the programmer to count in the units that are specified by the length of the address field. A 24-bit address field specifies bits; an integer in this field counts bits. A 19-bit field specifies half words so an integer here counts half words. An 18-bit field specifies full words, so an integer here counts full words.

Consider the following instruction.

C+I, $3, 13

The environment is determined by the operation C+I (Add Immediate to Count). This instruction has an 18-bit address field, so an integer is inserted with its unit bit in bit 17. This is equivalent to

C+I, $3, 13.0

and the integer can be considered to count full words. However, the same integer in the following instruction

V+I, $3, 13

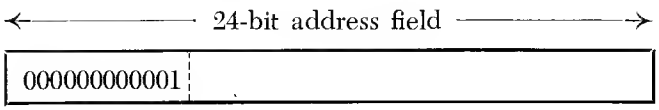
has a different meaning. Here the v+I instruction has a 19-bit address, and the integer inserted in this field is equivalent to 13 half words or location 6, bit position 32. This is the same as writing

V+I, $3, 6.32

The use of an integer to express an address requires special care on the part of the programmer since the size of the address field determines the interpretation of the integer. However, the integer is often the most desirable form of address specification, and simpler to use than a STRAP bit address. One such case is immediate addressing.

LI(BU, 12, 8), 1

The Load Immediate instruction above specifies, through its data description, a 12-bit address field. The integer address, in this case 1, is inserted as an integer in this 12-bit field. Thus, the instruction will be compiled by STRAP:



12-bit sub-field

The same Load Immediate instruction could be written with a STRAP bit address specification as follows:

LI(BU, 12, 8), 64.0

The two statements are equivalent, but the one written with the integer address is clearly the more desirable from the standpoint of simplicity of coding and recognition of original meaning when reviewing the statement at a later date.

PROGRAMMER SYMBOLS

A programmer symbol can be any sequence of 128 or fewer alphabetic and numeric characters that conform to the following conditions:

1. It contains only alphameric characters. This example

   THISISALONGNAME2SHOWTHATSTRAP NAMESCANBESENTENCES

   is a legal programmer symbol. This example

   A*B

   is not a legal symbol.

2. The first character is specifically alphabetic.

   6ALPHABET

   is not allowed, but

   A123456

   is perfectly acceptable.

3. It appears in the name field of a STRAP statement at some point in the program, at which time it is "defined" and is assigned a value that is either a standard binary bit address or an integer.

   BEGIN L(BU, 8, 8), A123456

   The symbol BEGIN is assigned a standard binary bit address which is equal to the value of the location counter within STRAP at the time this Load instruction is encountered in the code. The STRAP location counter always contains a 24-bit standard binary bit address.
   In the following case

   EIGHT SYN, 8

   the symbol EIGHT is assigned the value of the integer 8 through use of the pseudo operation Synonym (see Pseudo Operations).

Symbols that name instructions are automatically assigned data descriptions by STRAP. Specifically, an instruction-naming symbol is given a field length equal to the length of the particular instruction named (that is either 32 or 64 bits), a byte size of 8 and a use mode of binary unsigned (BU).

An integer in a programmer symbolized field is always converted to binary. An integer is limited in length to the length of the field in which it is to be inserted. An integer that cannot be expressed in 24 binary bits cannot be symbolized.

A programmer symbolized field is a field that may contain programmer symbols or system symbols. Of the fields shown in the instruction formats previously illustrated, all may contain programmer symbols except the operation field and the mode field of a data description.

SYSTEM SYMBOLS

System symbols are symbols whose values are fixed in the compiler. They are identified in programmer symbolized fields by the appearance of the special prefix character $ (which, as one of the non-alphameric characters, can never appear in a programmer symbol), followed by seven or fewer alphabetic or numeric characters. System symbols may appear in arithmetic expressions in programmer symbolized fields where, in cases where restrictions apply, they can be considered the same as numeric entries because their values are *immediately* available to the compiler.

All system symbols that represent the addresses of special registers in storage ($AOC, the All Ones

Counter) or special bits in storage ($LC, the Lost Carry indicator) are bit addresses. All others are real numbers.

The appearance of the $ character alone makes for a special system symbol that provides a standardized substitute in place of a name for the current statement. That is, the character $ is a bit address which, in any particular statement where it appears, functions as if it had been defined by being written in the name field of that statement. Because it represents the value of the location counter when the instruction is encountered by the compiler (if the instruction actually compiles space in the program), the appearance of the $ as follows:

```
B, $-2.
```

means "Branch to the instruction which begins two full words before this branch instruction." In another illustration:

```
B, $+.32
```

the meaning is "Branch to the next instruction," effectively a "no operation."

Another special use of the $ character is to prefix any operation code in this manner: $OP. This directs the compiler to suppress any error indications that arise in connection with the compilation of this statement.

STRAP assigns a dds to every system symbol. The system symbols can be classified in five groups. These are:

1. Index Register Symbols. The system symbols $0 through $15, or $X0 through $X15, represent index registers 0 through 15, addresses 16.0 through 31.0 in STRETCH storage. The advantage of using a system symbol is that STRAP always compiles the correct value, regardless of the size of the field in which the symbol is written. Therefore, in the instruction

```
*+(N),  ABLE($5)
```

the index specification involving the system symbol $5 directs STRAP to correctly compile the binary integer 5 in the 4-bit index subfield. In a similar fashion, STRAP correctly interprets the system symbol when used as an address as in

```
ST(BU),$X5
```

and compiles the standard binary bit address 21.0 in the address field of the Store instruction.

2. Special Register Symbols. The names of all the special registers in the 7030 computer are listed below, along with the system symbol for addressing each register, and the bit address assigned to each system symbol. STRAP also assigns a data description to each symbol with a use mode of binary unsigned (BU), a byte size of 8 and a field length equal to the length of the register. When a system symbol for a special register appears in the principal address field of a VFL instruction, no data description need be explicitly written out in that instruction.

NAME	MNEMONIC	BIT ADDRESS
Word number zero	$Z	0.0
Interval timer	$IT	1.0
Time clock	$TC	1.28
Interruption address	$IA	2.0
Upper boundary	$UB	3.0
Lower boundary	$LB	3.32
Boundary control	$BC	3.57
Maintenance bits	$MB	4.32
Channel address	$CA	5.12
Other CPU	$CPU	6.0
Left zeros count	$LZC	7.17
All ones count	$AOC	7.44
Left half of accumulator	$L	8.0
Right half of accumulator	$R	9.0
Sign byte	$SB	10.0
Indicator register	$IND	11.0
Mask	$MASK	12.20
Remainder register	$RM	13.0
Factor register	$FT	14.0
Transit register	$TR	15.0

A use of the system symbol for the Indicator Register is illustrated in the following instruction:

```
L, $IND
```

No data description need be explicitly written in the Load instruction because the dds (BU, 64, 8) has been attached to the system symbol. This instruction is thus complied by STRAP to mean, "Load the contents of the entire 64-bit Indicator Register into the right half of the accumulator at zero offset.".

3. Indicator Bit Symbols. The complete list of the system symbols for the indicator bits are listed in Appendix A. Each system symbol, when prefaced with a dollar sign and placed in a programmer symbolized field, will represent the correct bit position in word 11 of the indicator named.

The system symbols for the indicator bits are also used as part of the mnemonic for the Branch on Indicator instruction. In this usage, however, the $ is not required. The mnemonic for this instruction is composed of the B (representing Branch) followed by the system symbol for the indicator being interrogated minus the dollar sign. Thus, BXH is the operation mnemonic for Branch on Index High, while BXVGZ is the operation mnemonic for the 7030 instruction Branch on Index Value Greater Than Zero.

All the system symbols in classes 1, 2, and 3 are

bit addresses and are assigned standard data descriptions with mode BU, byte size 8 and a field length equal to the length of the particular register or bit.

4. Input-Output Address Symbols. Since the actual numeric addresses which are to identify particular I/O units and channels may be chosen arbitrarily, system symbols that represent integers are provided by STRAP for use in addressing I/O equipment. The numeric values of symbols in this class, unlike all other system symbols, may vary from one installation to another in order that RDR, for example, may represent the card reader channel address independently of what that address, in any particular installation, may be. The I/O system symbols are:

SYMBOL	MEANING
$PCH	Punch (Channel Address)
$PRT	Printer (Channel Address)
$RDR	Reader (Channel Address)
$DISK	Disk Unit (Channel Address)
	Note: The arcs of a disk may be addressed by any legal symbolic integer expression that is evaluated by STRAP modulo 2^{12} to assure a valid arc address.
$CNSL	Console (Channel or Unit Address)
$TC0, TC1, ... TCK	Tape channels 0, 1, 2, ... K

If more than one punch, printer, console or any other input-output unit is attached to the computer, the same numbering system used in tape channel addresses should be adopted where $PRT=$PRT0 for example; thus one may have $PRT1, $PRT2, etc.

Thus, a programmer may write the following Write operation:

W, $PRT, CONTROL WORD1

STRAP will compile the correct 19-bit channel address for the printer at that installation.

5. Symbols For Mathematical Constants. Five mathematical constants, useful in many scientific and engineering problems, can be represented by system symbols. These system symbols and their values are:

SYMBOL	MATHEMATICAL CONSTANT
$E	e
$M	$\log_{10}e$
$N	$\log_e 2$
$PI	π
$INF	∞ (infinity)

These five symbols may only be used in a data field of a Data Definition pseudo operation where normalized floating point (N) has been specified in the use mode field of the dds. The following data definition pseudo op

CONSTANT DD(N), $PI

assigns the floating point equivalent of the quantity π to the symbol CONSTANT. When CONSTANT is used in the address of a STRETCH instruction such as

+, CONSTANT

the normalized floating point data description is invoked and the full word floating point equivalent of π is added into the accumulator.

Index modification of an address field is performed in standard fashion. The index register to be used is specified as a sub-field of the address field. The index field is a 4-bit field, enclosed in parentheses, immediately following the address expression. STRAP bit addresses, system symbols and programmer symbols that are defined as bit addresses are all legal entries in an index field.

In the case of a bit address entry, the period is assumed to line up at the right end of the field. Thus, when converted to binary, the rightmost six bits of the entry are truncated, as are the leftmost 14 bits.

System symbols are the simplest to use and act as if a bit address had been entered. All of the following entries are equivalent in an index subfied of an address: $4.32 = 4.0 = \$4 = 20.0 = 52.0$

and all are translated by STRAP to mean index register 4.

If an integer is written in the index field, the meaning is entirely different. The integer tells STRAP that the symbol in the address field proper has been defined as an array and the integer is addressing an element in that array. (See Data Reservation Pseudo Operation.)

In the case of progressive indexing in a VFL instruction, it is the index register specified within the address field that is stepped by the immediate address.

Offset Field

Offset fields are similar in content to address fields. STRAP bit addresses, integers, system symbols and programmer symbols are all acceptable entries in an offset field.

An offset field has a fixed length of 7 bits. Probably the most common entry for an offset specification is an integer. An integer specifies a count of the number of bits to offset a field from the right end of the accumulator. An integer entry is converted to a 24-bit

binary integer by STRAP and the rightmost 7 bits are placed in the offset field. If a programmer writes the statement

L(BU, 64), PAYROLLDEDUCTION, 5

STRAP assembles the instruction to mean load the 64-bit quantity found at symbolic location PAYROLLDEDUCTION into the accumulator offset 5 bits from the right end. Since the offset field can contain a maximum of 7 binary bits, the programmer can specify any offset from zero to 127. When specifying offsets of greater than 64 bits or one full word, it may be more convenient to begin counting bits from the left end of the double length accumulator. This can be easily done by using negative offsets. The offset field is unsigned, hence STRAP translates any negative entry to the two's complement. The 128 bits of the accumulator, proceeding from left to right, are referred to by the offsets 127, 126 . . . 0 or, alternatively, −1, −2, −3 . . . −128.

If an offset specification is a parameter in a program that may vary from time to time, it is helpful to use a programmer symbol in place of an integer.

```
INDENT   SYN, 4
         (intervening instructions)
         ST(BU, 24),WORD1,      INDENT
         +(BU, 24),WORD2,       INDENT
         L(BU, 24),WORDSUM,     INDENT
```

The programmer symbol INDENT in the example above, can be defined as an integer early in the program, in this case by the pseudo operation Synonym. If the programmer changes the SYN card that defines INDENT to

INDENT SYN, 5

and reassembles, all offsets specified by this particular symbol are changed in value to 5 as well.

In the case of the offset field, care must be exercised when using STRAP bit addresses, not integers as in address fields. The reason for this is twofold. First, the length of the offset field is fixed, so an integer always has the same meaning. The bit address is also handled according to a fixed rule, but the meaning is not immediately clear from its appearance in the instruction. Second, a bit address is not the natural means of expressing an offset, and it unnecessarily complicates the specification. A STRAP bit address here will be converted to a 24-bit binary integer and the rightmost 7 bits will be inserted in the offset field while the leftmost 17 bits are truncated. Any STRAP bit address expression that specifies an address above 1.63 will overflow the offset field when converted to binary and only the rightmost 7 bits will participate. This occurrence can yield unexpected results.

The likelihood of using a system symbol to specify an offset is even more remote, but legal nonetheless. As previously stated, a system symbol is equivalent to a numeric entry; specifying an offset by means of a system symbol that is defined as a bit address, such as $IT in this example:

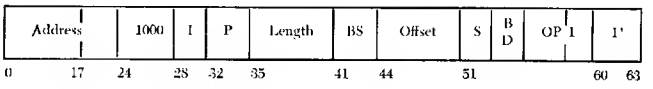
+(BU, 32), THISIS, $IT

is the same as writing 1.0 or specifying an offset of 64 bits.

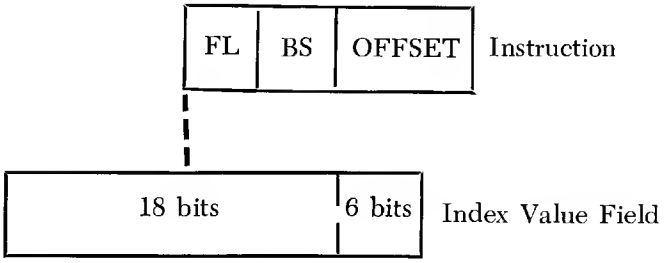
Index register specification is treated in the same fashion as an index modifier in an address field, except that the *modification can affect the field length and byte size as well as the offset.* The STRAP instruction format for VFL statements including data description as seen below

OP(M, FL, BS), A_{24}(I), OF_7(I')

does not hint at the relationship between field length, byte size and offset. The internal 7030 VFL instruction format,



the format into which a STRAP VFL instruction is translated, does show that the offset field is adjacent to the field length and byte size fields. The index modifier in the second half word treats all three fields together as one 16-bit field. For the modification process, the two fields are aligned as follows:



If the magnitude of the contents of the value field of the index register does not exceed 2^6, only the offset field can be modified. If the value field does exceed 2^6, the byte size *may* be affected (by a carry, for example). The diagram above shows how larger value fields *will* modify byte size and field length. This 7030 feature can provide very flexible and elaborate indexing of certain VFL instruction fields.

Pseudo Operations

Pseudo operations are operations created by STRAP-II to provide a simplified means of performing some special functions that are required in writing most pro-

grams. Definition of data, definition of symbols and setting a program origin are three examples of functions performed by pseudo operations.

Pseudo operations are not 7030 instructions; they do not exist in 7030 circuitry, but they resemble 7030 instructions in format. The general format for STRAP-II pseudo operations is

NAME POP(dds), A(I)

The pseudo operation code field appears first in the statement. The operation mnemonic is symbolized by POP. A complete list of mnemonics is given in Appendix B. A dds, if appropriate, appears as a sub-field of the operation field and is enclosed in parentheses.

The address field may contain a wide variety of entries that are not always addresses in the strict sense of the word. Some addresses can include index register specifications.

1. PRNID—Print ID

PRNID, XXXXXXXXXXXXX...XX

Normally, the first statement to appear in a program is the PRNID pseudo operation. The appearance of this statement instructs the assembly program to write immediately the entire contents of this card block on the output tape. PRNID provides a means of heading the assembly listing with such information as the problem name, programmer, and so on. A typical PRNID statement might be

PRNID, BCD CONVERSION ROUTINE BY JOE ZILCH

If a PRNID appears in the middle of a program, it will appear both at the beginning of the listing and at the point where it actually appeared in the code. When several PRNID statements appear in one program, they are listed sequentially in one group at the top of the listing and each one is listed in its appropriate place in the program. The practice of writing all PRNID statements at the beginning of the listing is useful, for example, when a program being assembled is composed of many subroutines, and each subroutine begins with a PRNID statement. The PRNID's, when they appear at the top of the listing, will form an index of the names of the subroutines included in that assembly.

A very long message may be written following a PRNID; if the message overflows the card, a continuation card or cards may be used. An alternate spelling of the mnemonic, PRINID, is also accepted by STRAP-II.

2. PUNID—Punch ID

PUNID, XXXXXXXX

PUNID fulfills the same basic function as PRNID except that the identifying information is punched on the binary cards produced by STRAP-II. The assembly program takes the first 8 characters following the comma that terminates the operation field, and punches them in columns 73-80 of every binary card produced as output of that assembly. The following statement

PUNID, IBMSINE1

causes the characters IBMSINE1 to be punched in the last 8 columns of each binary card produced in that assembly.

The identifying characters represented by x's above may be any legal card code characters except the ; and '. Every assembly must contain a PUNID statement or the binary cards will contain no identification other than the time clock setting as described under STRAP-II Punched Output.

3. SLC—Set Location Counter

A SLC, Y

In normal assembly operations, cards are read in sequence and the number of bits needed for each instruction or piece of data is added to a location counter maintained by STRAP to aid in the assigning of addresses to instructions and data. A principle of rounding upward is followed, guaranteeing that an instruction, value, count or refill will begin exactly at a half-word address, and that index words, control words and floating point data will begin only at full word addresses.

The SLC pseudo operation provides a means of setting the assembly location counter to any value at any point in the code, thus giving the programmer complete control over the location of his code. SLC resets the location counter to the value of the bit address Y. The next instruction will be compiled at this address, subject only to rounding upwards conventions. Following an SLC, the location counter is advanced once more in normal fashion until another SLC card resets it.

Y must be a bit address expression, either numeric or symbolic, whose value is positive. If an integer is specified in this field, it is treated as an integer in a 24-bit address field, i.e., it is interpreted as specifying a number of bits. Subject to this interpretation, it is evaluated correctly, but an error indication is given on the listing.

Any symbol in the name field will be effectively ignored, but will be entered in the symbol table.

If the following statement appeared in a program:

SLC, 100.32

it would cause the STRAP location counter to be reset to 100.32. If the instruction following the SLC were a VFL instruction, it would be compiled at 100.32. If it were a floating point data word, it would be compiled at 101.0.

4. XW—Index Word

XW, VALUE, COUNT, REFILL, FLAG

The location counter is rounded up to the next full word if it is not already at a full word address. The contents of the four fields following the operation are compiled in an index word format. The quantity represented by the symbol VALUE is compiled in bits 0-24 of the full word compiled. COUNT is compiled in bits 28-45 of this word and REFILL is compiled in bits 46-63. FLAG denotes the index word field composed of bits 25, 26 and 27. An expression in the flag field of an XW statement is therefore evaluated modulo 2^3.

If the following statement were encountered by STRAP in a program:

XW, 1001.50, TOTAL, XWORD2, 4

a full word would be compiled in the format of an index word with 1001.50 in the value field, the quantity symbolized by the programmer symbol TOTAL in the count field and the quantity symbolized by XWORD2 in the refill field, all converted to binary of course. The 4 is interpreted as the octal integer 4 in the three bit flag field, which turns on the index flag bit in the index word compiled.

Note: Bit 24, the 25th bit in the word compiled, is assumed to be the sign bit for the value field. All the other fields are unsigned; a negative sign is interpreted in two's complement form in the usual way.

5. VF—Value Field

VF, VALUE

The location counter is rounded to the nearest half-word if it is not already at a half-word address. The quantity symbolized by VALUE is compiled in bits 0-24 of the next half word (24 bits plus sign). The location counter stands at bit 25 at the end of the operation.

6. CF—Count Field

CF, COUNT

The location counter is rounded to the next half-word if necessary. The quantity symbolized by COUNT is complied as an 18 bit integer in bits 0-17. The location counter stands at bit 18 at the end of the operation.

7. RF—Refill Field

RF, REFILL

The pseudo operation is treated in exactly the same fashion as CF, except the word refill should be substituted for the word count.

Note: The last four pseudo operations defined above are given data descriptions by the compiler, and, therefore, cannot be written by the programmer. Specifically, the index words or elements created by these orders have had the following data descriptions affixed automatically, and cannot be overruled in the pseudo operation statement:

Operation	Data Description
XW	(BU)
VF	(B, 25)
CF or RF	(BU, 18)

8. CW—Control Word

CW(OP), ADDRESS, COUNT, CHAIN ADDRESS

The pseudo operation CW is similar in function to XW. CW creates a full word in storage, but in the format of an I/O control word. The location counter is first rounded up to a full word address unless it is already at a full word address. The quantity represented by the symbol ADDRESS is compiled in the first 18 bits of the full word created. COUNT is compiled in bits 28-45 and CHAIN ADDRESS is compiled in 46-63.

In a control word the flag bits (bits 25-27) are the chain flag, the multiple flag and the skip flag, in that order. Each of these bits may be set to zero or one and each of the combinations of the setting of these bits causes certain variations in reading and writing operations. STRAP-II defines 8 pseudo operations to specify all combinations of the three flag bits. The pseudo operation names indicate the type of I/O operation they specify. These pseudo operation mnemonics are written as a secondary operation in the CW statement, i.e., the mnemonics are written in parentheses immediately following CW. The 8 secondary pseudo operations are:

		Chain Bit	Multiple Bit	Skip Flag
CR	Count Within Record"	0	0	0
CCR	"Chain Counts Within Record"	1	0	0
CD	"Count Disregarding Record"	0	1	0
CDSC	"Count Disregarding Record, Skip and Chain"	1	1	0
SCR	"Skip, Count Within Record"	0	0	1
SCCR	"Skip, Chain Counts Within Record"	1	0	1
SCD	"Skip, Count, Disregarding Record"	0	1	1
SCDSC	"Skip, Count, Disregarding Record, Skip and Chain"	1	1	1

CW is assigned a data description of (BU, 64, 8).

9. DD—Data Definition

DD(dds), D, D', D'', ...

The Data Definition pseudo operation provides the programmer with the basic method of entering and

defining data. The dds in the operation field is identical in form and content to that previously described when writing a 7030 instruction, and must be written in every DD statement. Thus, a data description may be attached to a symbol at the point of definition of the symbol, or it may be written as a part of an instruction referring to the symbol.

When the data description is given by a DD statement (or other data defining pseudo operation), this description is invoked whenever the symbol appearing in the name field of the defining pseudo operation is used in the principal address field of a 7030 instruction. A description set down at the point of definition of the symbol is overruled by a data description appearing in the 7030 instruction that references the symbol. Whenever overruling occurs, the *entire* data description specified in the defining pseudo operation is overruled. Overruling applies only to the instruction at hand. Thus, the 7030 instruction

+(BU), SOMEMORE

explicitly specifies a data description of binary unsigned, field length 64 and byte size 8 (field length and byte size derived from null field conventions) to be compiled with this statement, regardless of the data description written in the statement where SOMEMORE was defined.

The address fields D, D', etc. shown in the general format above represent separate numeric entries which the programmer wishes defined by STRAP and converted to one of several 7030 internal forms. Several numeric entries may be written in one DD statement, separated by commas. D fields are signed fields (if use mode, B, D, N or U is given, of course). If no sign is written, the positive sign is assumed. The fields are converted and allocated storage sequentially as separate pieces of data, each having the data description specified. If too many D fields are written to fit on one card, continuation cards may be used to extend the statement field of the DD pseudo operation. If a symbol appears in the name field, it is attached only to the first piece of data compiled. When one wishes to name each of the entries, each must be presented in a separate DD statement with its own name.

Programmer symbols may not appear in the address field of a DD statement. (Pseudo operations VF or EXT may be used for this purpose.) It will be seen later that various letters have fixed meanings when they appear in a D field that are not subject to programmer control. Bit addresses, similarly, are not permitted in a D field. STRAP-II always assumes a numeric entry is written in the decimal radix, whether it is encountered in a pseudo operation or a 7030 statement. In a DD statement then, the programmer need specify only the form to which he wishes his data entry converted. This is accomplished by the use mode in the data description. All seven use modes—N, U, B, BU, D, DU, and P are all acceptable in a DD statement.

If use mode N is specified in a DD statement, as in

FLOATIT DD(N), 1000

the data entry 1000 is converted to its normalized floating point equivalent (in STRAP format) by STRAP-II, and placed in the full word storage location henceforth symbolically referred to as FLOATIT. Note that DD conforms to the normal STRAP rounding upward conventions. If use mode U had been specified in the dds, 1000 would have been converted to floating point in the same fashion, but not normalized.

Use mode B converts the numeric entry from decimal to binary. The sign byte is the low order byte of the converted number, equal in size to the byte size specified in the dds. The converted entry is placed in a field equal in length to the number of bits specified in the field length of the dds. If the field length specifies a field that is too small to contain the converted entry, the number is inserted in the field with the unit position aligned with the rightmost bit. Any high order bits that will not fit in the field are discarded. No rounding up of the location counter takes place. The field length specified is added to the current setting of the location counter and the numeric entry is converted and inserted in this field.

Use mode BU is essentially the same as B except that the entry is considered to be unsigned, and no sign byte is created. The entry is converted and inserted in a field of the length specified in the dds. The byte size specified has no effect on the conversion since an unsigned operation has been called for and no sign byte is compiled.

When use mode D is specified, a character-by-character type of conversion is called for, wherein each decimal digit in the numeric entry is converted to the four bit binary coded decimal form. If the byte size specified in the dds is greater than 4, high order zeros are added. If the byte size requested is less than 4, truncation occurs.

If use mode DU is specified, the conversion process is the same. However, no sign byte is compiled as none is required for the unsigned decimal mode.

To illustrate the differences between the binary and decimal modes, consider the following STRAP statements and the resulting fields compiled in storage:

STRAP Statement	Field Compiled
DD(BU, 4, 1), 1	0001
DD(DU, 12, 4), 12	000000010010

The P mode references the dds in another statement where the use mode must be N, U, B, BU, D or DU. Once the reference is made, the conversion performed by STRAP proceeds according to the rules already outlined.

To enter alphabetic information by means of a DD statement, a special entry mode subfield must be written, enclosed in parentheses immediately before the operation code as shown in this general format:

(EM)DD(dds), D

There are 4 entry modes available for use in entering alphabetic or alphanumeric messages. Each entry mode serves two functions; it tells STRAP that a message is being entered, and it describes the character set that is being used and prescribes the type of conversion that is required. When alphabetic information is specified, only one entry per DD statement is permitted.

(1) Entry mode A signals the appearance of a message composed only of characters drawn from the standard IBM BCD character set. If byte size 6 is specified in the data description, the characters are converted to the 6-bit IBM tape BCD format. If byte size 8 is given, 2 leading zeros are added to each 6-bit byte during the conversion process.

(2) IQS tells STRAP that the characters in the message are drawn from the set appropriate to the 7030 console typewriter, or Inquiry Station, and are to be converted to their 8-bit binary equivalents.

(3) P specifies that each character in the source language is one of the 120 members of the extended character set known as ECS 120. STRAP converts each character to its 8-bit equivalent.

(4) CC is the mnemonic for card code, and delineates that set of characters known as IBM card code characters. These characters are converted to 12-bit bytes, where each byte reflects the multiple punch actually read in the appropriate card column.

In the data description, the importance of the use mode and field length are deferred in that they cannot affect the conversion of the alphabetic characters but they do play an active role at a later time when another 7030 instruction refers to this alphabetic data and does not overrule the implied dds. The byte size, however, does affect the conversion of A characters but is ignored when any other entry mode is written.

If the following statement were encountered by STRAP-II:

(AQ)DD(BU, 60, 6), DONT PANIC Q

the compiler interprets the A entry mode to mean that the alphabetic data entry on this card is composed of BCD characters which are to be converted to IBM tape BCD format. The second character in the entry mode field is known as the end-of-statement character. Its presence instructs STRAP to perform the desired conversion until this character is reached in the message. The end-of-statement character may be any legal card code character except), ' (8-4), ;(11-0), and blank. This character is not compiled. Blanks that appear in the message are retained, and converted and stored correctly. A blank between the comma that marks the end of the operation field and the first alphabetic character is converted.

If the byte size specified is greater than 6, leading zeros are supplied by STRAP. If the byte size is less than 6, leading bits are truncated.

If IQS entry mode is specified, the conversion process is quite similar except that the characters are converted to the 8-bit inquiry station code. When the byte size specified is greater than 8, leading zeros are inserted; when the byte size is less than 8, leading bits are truncated. Note that in a DD statement, the byte size of converted characters may range from one through 12, as specified in the dds. However, the byte size in a 7030 statement may range from 1 through 8 because the byte size field is restricted to 3 bits in length. Therefore, byte size is treated modulo 8 by STRAP.

10. DDI—Data Definition Immediate

ANYNAME DDI(dds), D

The DDI pseudo operation performs the same basic function as DD, that is, it provides the mechanism for entering and converting data. In the case of DDI, the data in question is specifically intended for use as an immediate operand in a 7030 immediate instruction.

More specifically, DDI is the only convenient method for compiling decimal information in the address field of an immediate instruction. Data in an immediate address is *always* converted to binary, never to decimal, regardless of the use mode specified in the data description.

The data entry in a DDI statement is converted according to the use mode specified in the dds. The resulting field, which cannot exceed 24 bits in length (if it does, high order bits are lost), is inserted, *right justified*, in a 24 bit field in the STRAP symbol table. The field length specified in the dds is ignored at this point. When a 7030 immediate operation references this data through the symbol that appears as the name of the DDI statement, a field of the length specified in the implied dds or the overruling dds (if one is given) is extracted from the right end of the appropriate symbol table entry and is inserted *left justified* in the instruction address field.

In the following example,

```
IDATA   DDI(DU, 4, 4), 4
        LI, IDATA
```

the converted field created in the symbol table is

000000000000000000000100

while the 24-bit address field of the Load Immediate instruction will be compiled as follows:

010000000000000000000000

If the Load Immediate instruction had contained an overruling dds, such as

```
LI(DU, 8, 4), IDATA
```

the address field, after compilation would contain the following: 000001000000000000000000

If a signed use mode is given, such as

```
LI(D, 8, 4), –IDATA
```

then the symbol table entry would be

000000000000000000000100

and the instruction address field would be compiled as follows: 010010000000000000000000

If the length of the converted data entry is greater than the field length specified, high order bits (from the left) are truncated before insertion into the address field. Only the decimal or binary use modes, and the P mode of course, are legal in a DDI statement. The floating point use modes are not appropriate in immediate addressing, and hence are not acceptable. Any entry mode that is legal in a DD statement, including the alphabetic entry mode, is accepted in a DDI statement as well. If the field length is null in the specified dds, 24 is assumed by STRAP-II.

In summation, DDI is purely definitive in character; it compiles no space or binary output in storage. Data is converted and entered only in the symbol table. Data so defined that is referenced symbolically by a 7030 instruction is also inserted in the address field of that instruction.

11. SYN–Synonym

```
ANYNAME   SYN(dds), Y
```

The pseudo operation SYN provides another mechanism for defining a symbol in terms of an integer, a bit address or another symbol which is eventually defined as an integer or bit address.

When one writes

```
A   SYN, 6
```

the meaning of the newly defined symbol A is that whenever A is written in the program, the effect is the same as if 6 had been written. The meaning of SYN is always one of exact substitution.

The entry in the address field of the SYN statement is converted to binary and inserted right justified in a 24-bit field in the symbol table. In this process, SYN is similar to DDI, in that neither pseudo operation compiles space in storage. SYN statements may have their own data description; any dds that appears in a SYN is attached to the symbol in the name field, but in no way affects the conversion of the entry in the address field. When a 7030 instruction references the symbolic name of a SYN statement, the dds attached to that symbol is invoked as in DD. If no dds is given in a SYN statement, none is attached to the symbolic name. Then a dds must be explicitly written in a 7030 instruction that references a symbol defined by such a SYN statement.

Index registers may be attached to the expression appearing in the address field of a SYN statement. Thus, in the SYN statement:

```
A   SYN, B($3)
```

the index register specification it attached to the address expression, so that, the 7030 instruction

```
+(N), A
```

is synonymous with

```
+(N), B($3)
```

If an index register is specified in the principal address field of the instruction proper, it overrules any other index register specification for that instruction only. In the above example, if the normalized floating point add instruction had been written

```
+(N), A($6)
```

this would be synonymous with

```
+(N), B($6)
```

A circular definition may arise through the use of a sequence of SYN cards, as in the following example:

```
A SYN, B
B SYN, C
C SYN, A
```

All symbols in such a sequence are assigned a value of 0 by STRAP.

12. DR–Data Reservation, DRZ-Data Reservation and Set to Zero

```
A   DR(dds), N
```

A DR reserves storage space for data. The pseudo operation causes N fields of the kind described in the data description to be reserved; that is, the location counter is skipped forward a quantity in bits equal to the product of N and the field length specified in the dds. Any symbol A appearing in the name field of a DR statement is attached to the first field reserved, as is the data description. Thereafter, whenever A appears as the principal address in a 7030 instruction,

this dds is invoked in the same manner as with DD and DDI statements. Thus:

```
SAVE  DR(BU, 8, 8), 10
```

reserves 10 8-bit fields (skips the location counter forward 80 bits). The dds (BU, 8, 8) is attached to SAVE. SAVE is attached to the first 8-bit field reserved.

When either one of the floating point use modes is given in the data description, the floating point data block being reserved is forced to begin at a full word address. STRAP will automatically round the location counter up to the next full word address to accomplish this, thereby insuring that each floating point data word will begin at a full word address.

If no dds is given, the symbol appearing in the name field is assigned the normalized floating point use mode by STRAP.

By appending a Z to the DR mnemonic, a slightly different pseudo operation, Data Reservation and Set to Zero, is formed. This operation is identical to DR, but it performs the additional function of setting all reserved fields to zero. DR reserves fields but makes no attempt to clear them to zero.

DR can also define arrays. (See Section II).

13. END—End

```
END, Y
```

A card containing the pseudo operation code END signals the end of an assembly. Therefore, *an* END *card must appear as the last card of every symbolic program deck.* When STRAP recognizes an END card, it punches out a branch card with an address Y. This branch card is included as the last card of the binary output deck produced by STRAP. When the binary deck is loaded, the branch card causes control to be transferred to the instruction located at Y.

Since the instruction located at Y will be the first instruction in the program to be executed, Y usually specifies the location of the first instruction in a program. This use of END is illustrated in the following example.

```
        SLC, 1050.
BEGIN   L(BU, 24), DATA
        (intervening code)
        END, BEGIN
```

Of course, the END statement does not have to address the first instruction in a program. The programmer is free to select any instruction he wishes to be executed first. If the END address is a programmer symbol, STRAP correctly substitutes the STRAP binary bit address equivalent. If the address is a numeric entry, the programmer is cautioned that the address follows the rules of any 24-bit address field. An integer written in this field is interpreted as a number of bits. A bit address will be compiled correctly, so care must be taken to include the period unless an integer expression is specifically intended.

Any symbol appearing in the name field is effectively ignored by STRAP, but the symbol is placed in the symbol table.

STRAP-II Output Listing

Basically, the output listing produced by STRAP-II contains two types of information. On the right half of the page, each STRAP statement is reproduced as it was punched on the symbolic input card. Thus, each line of the listing represents one symbolic card. On the left half of the page, the location assigned each statement is displayed in octal, followed by an octal-hex representation of the compiled information. A sample listing appears in Figure 3.

The octal-hex representation is one which, as the name implies, uses two different radices to represent each half-word instruction compiled by STRAP-II. The first 24 bits of the half word are displayed in octal, with a period supplied between the sixth and seventh octal character (between the 18th and 19th bits in binary) to facilitate reading STRAP bit addresses. An economy can be effected by representing the last 8 binary bits of the half-word by 2 hexadecimal characters. (Any 4-bit binary integer, that is any number from $0\text{-}15_{10}$, can be represented by one of the hexadecimal characters 0-9, A-F. Thus,

$0001_2 = 1_{16}$ $\quad$ $1010_2 = A_{16}$

$1001_2 = 9_{16}$ $\quad$ $1111_2 = F_{16}$

The subscript 2 refers to the binary radix, while 16 refers to hexademical.) If a full word instruction has been compiled, two half-word octal-hex expressions are used.

The octal-hex notation is used only for displaying compiled instructions. At least four other print formats are available:

1. *Floating Point.* When a Data Definition statement with a floating point use mode is specified, the compiled data entry is printed in octal but it is separated into the components of the 7030 floating point format- exponent, exponent sign, fraction, fraction sign and data flags. See lines 14 and 15 in Figure 3.
2. *Index Word.* When the XW pseudo operation is employed to create storage elements in the format of 7030 index words, the printed display of the compiled information is clearly divided into the four fields comprising the index word-value field plus sign, index flag and two unused bits, count field and refill field. See line 17 in Figure 3.

```
                    TIME CLOCK        011000101
                                                        PAGE              1
      LOCATION              BINARY OUTPUT          NAME      STATEMENT
000077.00 LOWER MEMORY BOUND
000115.00 UPPER MEMORY BOUND
 1- 000100.00                                                 SLC,64.
 2- 017777.00+         +00000000                      NEXT     SYN,(8)17777.0
 3- 000100.00     000112.16 10                       ABLE     LX,7,INDEX                -LOAD INDEX
 4- 000100.40     000113.20 80 000000.20 50                   L(BU),BAKER
 5- 000101.40     000000.10 87 204000.20 D0          CHARLIE  ST(BU,4)(V+IC),.8($7)
 6- 000102.40     000101.70 40                               BZXCZ,CHARLIE
 7- 000103.00     000107.00 60                               L(N),DOG
 8- 000103.40     000110.00 20                               +(N),DOG+1.
 9- 000104.00     000111.00 E0                               ST(N),FOX
10- 000104.40     000113.34 80 030000.20 50                  L(BU,24),SOME VERY LONG NAME
11- 000105.40     000113.64 80 030000.20 D0                  ST(BU,24),SOME OTHER LONG NAME      -TESTING LONG COMMENTS THAT
12-                                                          WILL CARRY OVER TO THE NEXT LINE
13- 000106.40     017777.10 00                               B,NEXT

14- 000107.00    0007+ 6777700000000000 +000     DOG       DD(N),28671X7,183007S7
15- 000110.00    0022+ 5453370000000000 +TUV
16- 000111.00   000001.00                         FOX       DR(BU),(1)
17- 000112.00   000113.00+ 000 000004 000000      INDEX     XW,ZEBRA,4
18- 000113.0    000000.20                         ZEBRA     DR(BU,4),(4)
19- 000113.20                     5703            BAKER     (8)DD(BU,12),5703
20- 000113.34                   00067777          SOME VERY LONG NAME
21-                                                          DD(BU,24),28671
22- 000113.64   000000.30                         SOME OTHER LONG NAME
23-                                                          DR(BU,24),(1)
24- 000114.14   000100.00                                    END,ABLE
    THIS ASSEMBLY REQUIRED       00000032 SECONDS
```

FIGURE 3.

3. *Octal.* Binary signed and unsigned data compiled via a DD statement are printed on the output listing in a straight octal format. See lines 19 or 20 in Figure 3.

4. *Decimal.* A decimal use mode in a DD statement cause the compiled data to be displayed in decimal.

Pseudo operations that do not cause any binary information to be compiled give rise to certain unique printing formats. DR compiles no binary information, so STRAP prints the number of words and bits reserved by the pseudo-operation as an octal bit address. SYN, on the other hand, can define a symbol in terms of either an integer value or a STRAP bit address value. When the symbol is defined as a bit address, an octal bit address equivalent is printed in the column where the location counter setting is usually displayed. If the symbol is defined as an integer, a straight octal representation of the converted integer is printed where all other compiled statements and data are shown. If the pseudo operation SLC is used, the contents of the location counter resulting from the appearance of the SLC are displayed in the usual column as an octal bit address.

Some additional information is supplied on the listing which will prove helpful to the programmer. The first item to appear on each assembly listing is a binary representation of the 7030 internal time clock when the assembly began. The time clock can be used for identification purposes. The time required to complete the assembly is displayed in seconds as the last item printed on the listing. Headings are also given over each column of information to clarify where Location, Binary Output, Name and Statement appear.

Also, at the beginning of the listing, four lists of symbols are supplied by STRAP. The first list is a tabulation of those programmer symbols that were not defined by the programmer, along with the definitions supplied by STRAP. The second list contains all programmer symbols that are defined by the programmer but are never referred to or used. The third and fourth lists contain those symbols that are multiply defined with contradictions and pseudo defined.

Immediately following the column headings, upper and lower storage bounds are printed as octal bit addresses. The boundaries for each program are determined by STRAP in the following way—the lower memory bound is the address of the full word in storage immediately preceding the first word used by the program, while the upper memory bound is the address of the full word in storage that immediately follows the last word used by the program.

The leftmost column on the STRAP-II listing contains line numbers—the printed lines on each page are numbered sequentially, beginning with 1. Each page is also numbered, and this number appears at the top of the page, just below the time clock display. STRAP can easily refer to any line of printed output by page and line number.

Certain error conditions can be detected by STRAP during compilation. At the completion of an assembly, then, STRAP can list error messages and in each message reference the statement by page, line number and field wherein the error occurred. Since many statements occupy more than one line on the listing (see lines 11 and 12 on the sample listing), an error message will reference only the *last* line occupied by the statement's binary output. Consult Appendix D for a complete list of error messages.

Five other error conditions, all caused by incorrect definition of programmer symbols, can be detected by STRAP and reported on the output listing by means of error flags. These flags are five or six character symbols that appear on the listing on the line immediately preceding the first line of the statement that contains the symbol erroneously defined. The five flags and the meaning of each are:

(1) UNDEF. An undefined symbol has been detected. STRAP has assigned to this symbol the bit address value of the first full word location following the highest full word used by the program in which this symbol appears. If several symbols are undefined, they are assigned sequential full word locations from this starting point, in the order in which they are encountered by STRAP.

(2) QUEST. A multiply defined symbol has been encountered. However, the definitions are not contradictory, that is, two or more definitions of the same programmer symbol have been found and all definitions assign the identical value to the symbol. This situation could occur in this sequence of instructions:

```
        SLC, 1000.0
SYMBOL  LI, ANOTHERSYMBOL
        +I, STILLANOTHER
SYMBOL  SYN, 1000.0
```

STRAP accepts the definition as legal and does assign the specified value. The appearance of the flag is to warn the programmer of the unnecessary multiple definition.

(3) MULTI. This flag signals a more serious case of multiple definition where the definitions are contradictory. If the following two statements were

found in a program

```
A SYN, 100.0
A SYN, 100.32
```

the MULTI error flag would appear on the output listing on the line immediately preceding the second SYN statement. When contradictory definitions occur, STRAP assigns the first value encountered and discards all subsequent definitions.

(4) PSEUDO. Pseudo definitions are often called circular definitions and are best illustrated by the illustrations below.

```
A SYN, B
B SYN, A+5
```

STRAP-II assigns a value of zero to A and a value of 5 to B.

(5) CONTAG. A contagious error occurs wherever a programmer symbol is defined in terms of another programmer symbol which has been erroneously defined in one of the four ways described above. In the following case

```
   SLC, 500.0
A SYN, 1000.0
A SYN, 500.0
B SYN, A
  L(N), B
  +(N), A
```

MULTI flag would appear on the listing on the line immediately preceding the Add statement, and the CONTAG flag would be found on the line preceding the Load statement. STRAP would assign the value 500.0 to A and B.

Because STRAP-II has provisions for very long programmer symbols and continuation cards, the symbolic listing of the contents of the input cards may extend over two or more lines. If the name of the statement is too long to fit in the name column, it extends into the statement column, and the remainder of the statement is printed on the next line. An illustration of this is found on line 20 and line 21 of the sample listing. Note that even though the statement uses two lines, the compiled binary information is printed on the first line. In another instance, the programmer may use a continuation card to append a very long comment to a statement. An example of a long comment forcing a format change in the listing is seen on lines 11 and 12 in the sample listing.

The reverse situation occurs when several D fields are written on one DD card or multiple statements are written on a card. Then the binary output will be spread over two or more lines, while the symbolic duplication of the input card appears on one line. Lines 14 and 15 illustrate a DD with more than one data entry.

STRAP II Punched Output

In addition to printed information, STRAP also punches column binary cards as part of the output of each assembly. Four types of column binary cards are punched.

Origin Card. The first card of every binary deck to be loaded into the 7030 via the standard loader program must be an origin card. Basically, the origin card contains an origin address, a checksum and up to 23 half-words of data and/or instructions. The origin address tells the loader where to start loading the compiled information that appears in column binary form on the origin card in columns 10 through 72. The origin address is taken from the SLC statement that is normally the first statement in any program following the PRNID and PUNID pseudo operations. The checksum permits the loader to check that the binary information on the card has been correctly loaded.

The complete format of the origin card is shown below. In the convention used to number card columns and rows, the first number specifies the card column—a number ranging from 1 through 80. The second number, separated from the column number by a period, is the row number. Here the card is considered to be divided into 12 rows—the row nearest the top of the card is row 0 and the row nearest the bottom of the card is row 11. For example 10.8 means column 10, row 8.

Card Column and Row	Use
1.0-1.11	Code column (origin card—1.9, 1.10, 1.11 punches)
2.0-2.11	Identification column (binary)
3.0-3.11	Sequence number (binary)
4.0-4.11	Checksum
5.0	A control-bit—0 if skipping, 1 if setting to zero
5.1	A control bit—0 if action is before card contents are loaded, 1 means after card contents are loaded.
5.2-5.11	Primary bit count
6.0-7.11	24-bit origin address
8.0-9.11	Secondary bit count
10.0-10.7	Not used
10.8-71.11	Up to 736 information bits
73.0-80.11	Identification (card code)—ignored by the loader

The additional fields seen in the format have the following uses:

1. Code column—this is a multiple punch code that tells the loader the type of card that is being loaded. For an origin card the code is a punch in 1.9, 1.10 and 1.11.

2. Identification column—12 bits of the 36-bit 7030 time clock ($TC) are punched in column 2 of every binary card produced by STRAP-II to identify each assembly. The setting of the clock at the start of each assembly is used. Column 2 will be ignored by the loader.

3. Sequence number—a binary number computed by STRAP to aid the loader in checking the sequence of cards being loaded. The first card is every deck punched by STRAP is given the sequence number 1, the second is given sequence number 2, etc.

4. Primary bit count—this 10-bit count tells the loader the number of bits of binary information (columns 10 through 72) that are to be loaded into 7030 storage, starting at the location specified by the origin. Any number from 0 to 748 can be specified. Bits not intended to be loaded are ignored by the loader.

5. Secondary bit count—this 24-bit count is inter-

preted by the loader in conjunction with the two control bits.

Bit5.0	Bit5.1	Meaning of Secondary Bit Count
0	0	Skip n bits before loading card contents
0	1	Skip n bits after loading card contents
1	0	Set n bits to zero before loading card contents
1	1	Set n bits to zero after loading card contents

Bit skipping or zeroing before loading is started at the origin address. Skipping and zeroing after loading is done starting with the bit location immediately following the last bit loaded from the origin card. Information for skipping or zeroing is determined from the pseudo operations DR and DRZ. If DR has been given, bit skipping is called for, while DRZ specifies setting bits to zero. The setting of control bit 5.1 is determined by the position of the DR or DRZ in the code. (See flow card below.)

6. Identification—STRAP punches in this field the card code characters specified in the last PUNID statement encountered.

Flow Card. A flow card contains 25 half-words of data in column binary form. This data is to be loaded in sequence with the data of the previous card loaded. The format of the flow card is:

Card Column and Row	Use
1.0-1.11	Code column (flow card—1.9 and 1.11 punches)
2.0-2.11	Identification number (binary)
3.0-3.11	Sequence number (binary)
4.0-4.11	Checksum
5.0-5.3	Not presently used
5.4-71.11	25 half—words of binary information
73.0-80.11	Identification field ignored by the loader

All columns reserved on a flow card to contain compiled data or instructions must be used. No primary bit count is provided for. All of these columns are read by the loader, and any that contain no punches are interpreted and loaded as if they contained zeros.

When a DR or DRZ immediately follows an SLC, the skipping or zeroing information can be placed on the origin card and the proper control bit set to accomplish the zeroing or skipping before loading the contents of the origin card (because the contents are instructions or data that follow the DR or DRZ in the program). Another situation that often occurs is when STRAP is constructing a flow card and a DR or DRZ is encountered before the data columns (5.4-7.11) are full. STRAP immediately changes the card to an origin card; now a primary bit count can be given so that instructions and data ready to be punched in the card can be loaded, but the remaining blank columns can be ignored. Now a control bit can be set so that the skipping or zeroing is done after the contents of the converted origin card are loaded.

PUNFUL Card. A PUNFUL card is a special type of flow card requested by the programmer through the PUNFUL pseudo operation. The format of the PUNFUL differs from the flow card in that all 80 columns of the card are used for column binary data or instructions.

A PUNFUL card cannot be loaded by the standard loader. In normal usage, PUNFUL cards containing constants or tables of data are placed behind flow cards. A TLB pseudo operation is positioned between the last instruction on the flow cards and the first PUNFUL card. The action of the TLB is to interrupt loading and give control to the problem program. At the appropriate point in the program, the programmer can load the PUNFUL cards under program control.

Branch Card. A branch card contains an address to which the loader transfers control. A branch card is produced as a result of STRAP encountering an END card or a TLB card. If no address is specified with the pseudo operation, control is transferred to the address given as the origin on the first origin card produced for the subject program.

The format of the branch card is:

Bits Assigned	Use
1.0-1.11	Code column (branch card—1.8, 1.9,
2.0-2.11	1.11 punches)
3.0-3.11	Identification number (binary)
4.0-4.11	Sequence number (binary)
5.0-5.11	Check sum
6.0-7.11	Not presently used 24-bit transfer address

The card before the branch card is often forced to be an origin card. As before with DR or DRZ, if the TLB or END is encountered when the flow card being composed does not have columns 5.4 through 71.11 filled the flow card is changed to an origin card. The next card will be the branch card.

Entry Mode

STRAP II always assumes that the characters appearing in the statement on a symbolic card are alphabetic or numeric. Furthermore, when the characters are numeric, STRAP assumes they are written in the decimal radix. Often it is much more convenient to write a numeric entry in another radix, such as octal or binary. In other cases there are other properties of the source language that the programmer would like to describe to STRAP. The facility in the STRAP language that allows the programmer to describe the source language is called the entry mode.

Within the data description field the use mode, field length and byte size describe characteristics of the data that determine the conversion of the data and its later use at execution time. These characteristics are therefore compiled along with the data. The entry mode, on the other hand, describes the form in which the data appears on the card and, therefore, need not be compiled. The entry mode may be employed in one of three ways:

Statement Entry Mode

An entry mode may be used to specify the properties of all data in a DD or DDI statement. When used in this fashion, it is enclosed in parentheses and appears immediately before the DD or DDI operation code in the operation field.

(EM)DD(dds), D, D', D'', ...

Note that DDI does not have the multiple entry facility.

When an entry mode is used in connection with the data of a DD or DDI statement, it may in this instance —but only in this instance—designate that alphabetic information is to be compiled. There are only two entry modes that fall into this category, the A entry mode and the IQS entry mode. These modes have already been discussed in the section concerning the DD pseudo operation.

Statement or Field Entry Modes

Some entry modes may be used to specify the properties of all the fields in a statement or to specify the properties of one specific field within the statement. One such entry mode is the F mode.

F ENTRY MODE. The F mode may appear only in DD or DDI statements where an unnormalized floating point or binary use mode has been specified. If the F mode is employed as a statement entry mode within such a statement, it is written enclosed in parentheses immediately before the operation code.

(F6)DD(BU), 12.36

In this case, the entry mode F implies that the data which follows are written in the decimal radix, are to be converted to binary, and may contain a decimal fraction portion. The integer following the F specifies the number of fractional binary bits that are desired to the right of the binary point following conversion. In the previous example, the fractional portion to the right of the binary point will be limited to 6 bits in length. The converted 6-bit fractional portion plus the integer portion will be right justified in the appropriate field (in this case a 64-bit field so leading zeros will be supplied by STRAP).

Conflicts between the field length specified and the F entry mode can arise where binary use mode has been written. If the converted data entry is too large to fit in the field requested, high order bits are discarded. Whenever the converted entry is smaller than the field size specified, the problem is less crucial. High order zeros are supplied.

In the case of unnormalized floating point DD statements, the rules governing the interpretation of the data and its conversion are identical to the handling of binary use mode statements except that the converted data entry is always inserted right justified in the standard fractional portion of the floating point format. The correct exponent, as determined by the location of the decimal point, is supplied by STRAP.

Entry mode F may also be used as a field entry mode, that is, it may be used to specify the properties of one particular field within a DD or DDI statement without influencing the treatment of any other field in the same statement. In everyday programming situations, it is common to write DD statements with several data entries in each statement. In this situation, it is often desirable to use different entry modes for each field. Thus, the programmer may write

DD(BU), (F6) 12.36, (F2) 187.5, (F8) 1005.679

Note that when the F entry mode is used as a field entry mode it is still enclosed in parentheses and ap-

pears first in the field. The meaning is the same as when it appears as a statement entry mode; however, that meaning applies only to the data entry in the field in which it appears.

Statement entry modes and field entry modes may both appear in the same statement. When there are contradictory properties by the statement and field entry modes, the field entry mode overrules for the case of the particular field on hand. Entry modes may not appear in a manner that cause parentheses within parentheses. In the following example:

(F6)DD(BU), 12.36, (F3) 166.3, 1776

the F6 entry mode rules for data entry fields one and three, while the F3 specification temporarily overrules the statement entry mode for the second data entry only.

RADIX SPECIFIER The radix specifier is another entry mode that may be used as a statement entry mode or a field entry mode. In any programmer symbolized field not enclosed by parentheses, numerical integers and bit addresses may be written in any radix from two through 16. The radix is specified by enclosing the appropriate decimal integer in parentheses and placing it either before the operation code if statement entry mode action is desired, or at some appropriate place in the field to which it refers when it is employed as a field entry mode. (Usually, but not always, the radix specifier is the first item to appear in the field.)

If used as a statement entry mode, the radix specified applies to the entire statement unless individual fields contain their own radix specifier, in which case the field entry mode overrules the statement entry mode for that field only. If used as a field entry mode, the radix applies to the entire field unless it is reset before the end of the field is reached. If no radix is specified, the base 10 is assumed.

1. (8)573–34+50 (all numbers are in octal)
2. (2) 11011011100011.111100 (bit address written in binary)
3. (5) SAM–342 (the symbol SAM is not affected by the radix, having been previously converted to binary. The integer 342 is written in the number system of the base 5.)
4. (8)7436.(10)60+9 (the full word portion of this bit address is written in octal, whereas the bit address portion and the integer 9 are written in decimal.)
5. (2)DD(B, 16, 8), (10)–972, 111011110 (the first D field is written in decimal, the second one is binary.)

The entry mode radix specifies the radix in which an integer is written on the card but says nothing about the one to which it is converted. At the completion of every STRAP statement, the radix is automatically reset to 10 and remains 10 for the following statement unless it is changed therein.

One note of caution applies to the use of radix 16. An address expression written in hexadecimal *must* begin with a numeric character.

Field Entry Mode

A final STRAP entry mode that is used as a field entry mode is the parenthetical integer entry mode. This mode permits any integer or pattern of bits to be stored in any bit position of an instruction or pseudo operation that produces binary output. The general format for this entry mode is:

$$(.n)A_{n+1}$$

The symbol .n represents the bit address of the rightmost bit of the field into which the integer or bit pattern is to be entered. The integer A_{n+1} is formed as an unsigned field, n+1 bits in length (because of the 7030 custom of addressing bits starting with zero), and inserted into the leftmost n+1 bits of the addressed instruction or data entry field by means of a logical OR type operation. Logical OR is used so that the parenthetical entry may be combined with the existing contents of the particular field addressed or with other parenthetical entries.

The field selected by the parenthetical integer entry mode may cross field lines within a statement as determined by the format of the statement. However, the parenthetical entry mode is not permitted to cross statement lines. The specification of the rightmost boundary of the addressed field via .n must therefore be less than or equal to 31 in a half word instruetion, or 63 in a full word instruction. Nevertheless, a maximum of 24 significant bits can be converted in a parenthetical entry. If necessary, zeros are added to expand to the desired length. When the bit address is specified as .n, the parenthetical integer expression is assigned a field length of n+1 and is evaluated modulo 2^{n+1}. All parenthetical fields are regarded as unsigned by STRAP, so that a negative number is compiled as the complement, re 2^{n+1} of the magnitude of the number.

In the following instruction:

E+I, (.8)41

the integer 41 is converted to binary and OR'ed into the first nine bits of the E+I instruction. In the case of an instruction, the position of the entry is determined by counting bits from the beginning of the instruction, starting with bit zero, no matter in which

subfield of the instruction the integer entry may be written. Thus, in the VFL instruction format, the parenthetical integer entry may be appended to the address field, as in this illustration:

+(BU), DATA(.23)4, 20

or it may follow the offset specification as in this illustration: +(BU), DATA, 20(.23)4

In either use the result would be the same. The rule is that the parenthetical entry must follow all other information in the field in which it does appear.

When a parenthetical integer entry mode appears in the D field of a DD statement, the .n specification names the rightmost bit position relative to the beginning of the field at hand, not relative to the beginning of the DD statement. In other words, the parenthetical field position is determined by counting bits from the previous comma forward. In DD statements with multiple data entries, one or many parenthetical entries may be appended to each such field. Again in the case of DD only, the .n specification is restricted to be less than or equal to the field length as given in the data description of the particular statement.

There is no limit on the number of consecutive parenthetical integer entries that may be written. Although one entry can conceivably be made to serve any single instruction or data field, is it often convenient to write several different integer entry specifications when one wishes to place numbers or patterns of bits in various positions within an instruction or data field.

This entry mode must appear in a statement that compiles space in storage. Therefore, this mode cannot be used in pseudo operations that give instructions to the compiler but result in no binary output (SCL, DDI, END, etc.). The parenthetical entry mode is a modification that may be appended to a D field or to any programmer symbolized field (or in place of such a field) which is not enclosed in parentheses. Thus, an index register specification in an address field may not contain this entry mode. One exception to this rule is permitted in DD statements only. Here, a parenthetical integer entry may be written in the data description field which is enclosed in parentheses. When so written as an appendage to the field length or byte size specification, but never as a modification of the use mode, the meaning is similar to that of a statement entry mode. That is, the parenthetical integer entry acts as if it had been appended to each of the D fields that follow in the DD statement. This unusual notation permits the insertion of a pattern of bits in every data entry in a multiple D field DD statement without the necessity of repeatedly writing the parenthetical entry in every field. In all other respects the parenthetical entry mode behaves exactly the same as it does when used as a field entry mode.

Parenthetical expressions may contain anything that goes in a normal address field (except bit addresses), but may not contain other information such as alphabetic messages or real numbers (see Rules for Entering Data) which are permitted in DD or DDI statements. If a programmer symbol is used as a parenthetical integer entry, any data description associated with this symbol has no effect on this particular usage of the symbol. All numbers that appear in a parenthetical field are converted to binary, never to decimal or floating point.

Radix designators are permitted in parenthetical OR fields, separated by commas from the bit address designation, and the two may be in any order. Thus, (.32, 8) or (8, .32) signifies a parenthetical integer entry follows that is written in the octal radix on the card and is to be inserted in the field whose rightmost boundary is bit position 32.

Examples:

1. L(BU), INFO(.50,8)17–JOE+(10)4203(4,.22)–33303(.60)1030
2. L(BU), INFO(7)(.30)1265(.20)(10)138–(6)43(.10)553

The first example is that of a VFL instruction with three consecutive parenthetical integer entry expressions appended to the address field. It is interesting to note that arithmetic between integers and programmer symbols is permitted in forming the integer entry (17_8–JOE+4203_{10}) and that when no radix is specified with a parenthetical entry, the current operative radix is continued. No attempt is made to reset to 10. The radix is assumed to be 10 if no radix has been previously specified in the field to which the parenthetical entry is appended, and if no radix has been specified as a statement entry mode.

The second example also illustrates 3 separate parenthetical integer entries in the address field. Of significance here is the fact that the radix need not be specified within the same set of parentheses as the bit address specification for the integer entry.

The radices which apply in the above examples are:

Example	Number	Radix
1	17	8
1	JOE	does not apply
1	4203	10
1	33303	4
1	1030	4
2	1265	7
2	138	10
2	43	6
2	553	6

All numbers that appear within parentheses are interpreted by STRAP as decimal numbers.

The Form of Data Entries in DD Statements

Any number written in a DD statement for conversion by STRAP must be capable of being expressed in 64 binary bits. This means that the largest fixed point quantity that can be converted by STRAP is equal to 2^{64} or 18, 446, 744, 073, 709, 551, 615 or 20 decimal digits.

The floating point data format is a special case. Here the numeric entry is always converted to a 48-bit fraction and an 11-bit exponent. Therefore, the only decimal quantities that can be expressed in 7030 floating point format must lie within the range 10^{-308} to 10^{308}.

Numeric entries in Data Definition statements may be written in a variety of formats. The two basic formats are the integer format, such as

982104

and the decimal fraction format, as in

–982104.2

These illustrations are written in the decimal radix. As previously described, an entry mode in the form of a radix specifier can be employed so that the programmer may write the data entry in one of several radices. If no sign is written, the number is assumed to be positive. If a BU or DU use mode is given and a sign is written, the sign is ignored by STRAP.

Some special characters may be appended to data entries to provide further flexibility in notation. It is often convenient to express a data entry as a number raised to some power of 10. The suffix letter E is used for this purpose, as in this example:

670.7E7

The meaning of E is to multiply the number that precedes it by the power of 10 expressed by the number that follows it. This number is always interpreted as a decimal integer. Thus, the above example is interpreted by STRAP to mean 670.7×10^7. The presence of E automatically implies that the entry is written in the decimal radix. If a floating point use mode is specified, both the E specification and the position of the decimal point affect the computation of the exponent.

Two other suffix characters are used for the insertion of specific fields.

SIGN BYTE ENTRY.

The letter s is used to enter information into the sign byte of signed data. Any integer that follows the s is interpreted by STRAP as an octal integer. It is converted to binary and inserted by means of a logical OR into any previously calculated sign byte.

The sign byte generated depends upon the byte size specified in the data description; its composition is illustrated by the following table.

Byte Size	Sign Byte	
1	S	
2	ST	
3	STU	
4	STUV	Z = zone bit
5	ZSTUV	S = sign bit
6	ZZSTUV	T, U, V = flag bits
7	ZZZSTUV	
8	ZZZZSTUV	

A byte size of 1 means that the sign byte is composed only of the sign bit; hence, an octal 1 will be ORed into the sign bit position and create a negative sign. If the specified byte size had been 4, the suffix s10 would be required to create a negative sign. Because the logical OR is used for the insertion, the sign byte sign position can be made negative by either a negative sign written with the numeric entry or by an s-type entry.

EXPONENT ENTRY.

The suffix letter x may be used if the programmer wishes to create his own exponent for a floating point data entry. The number following the x is interpreted by STRAP as a decimal integer and is converted to binary and compiled as the machine exponent of the floating point number to which it is attached. It overrules and replaces the exponent computed by STRAP in the conversion process, which is completely eradicated by the replacement process.

Complete Rules for DD Statements

The legal formats for entering data can be classified according to the use mode written in the data description field of the DD statement. Normally, an element listed in the general format may be omitted if it is not needed to specify the data.

The data entries in a DD statement are restricted to real numbers. Bit addresses would have no meaning here and are not allowed. In addition, programmer symbols are not permitted. In one special case, where normalized floating point has been specified in the data description, the system symbols for certain mathematical constants are accepted.

Arithmetic expressions, that is, combination of two or more numbers by means of addition, subtraction, multiplication and division to form one data entry, are permitted in all DD statements regardless of the use mode specified. Such arithmetic is specified using the standard FORTRAN symbols. The symbols available are addition (+), subtraction (–), multiplication (*)

and division (/). STRAP will perform the arithmetic and compile a single constant. Multiplications are performed first, proceeding from left to right, and then the additions and subtractions are completed.

STRAP does the necessary bookkeeping to insure that floating point data entries are always compiled at addressable full words; the location counter is rounded up to the nearest full word, if necessary, in order to accomplish this.

Normalized Floating Point

Format:

Name | DD(N), ±xx···xx.x···xxE±yyySn

The number is converted to a normalized floating binary number consisting of an 11-bit signed exponent, a 48-bit fraction, and a 4-bit sign byte. If no sign byte has been entered by means of an s, the sign preceding the number is used with the flag bits set to zero. If a different binary exponent is desired, it can be entered following an x, as follows:

Format:

Name DD(N), ±xx···xx.x···xxE±yyySnXzzz

Examples:

a. DD(N), 54.73 E 4
   54.73×10^4 is converted to floating binary. The sign bit is zero (= plus), and the flag bits are zero (i. e., entire sign byte is zero).

b. DD(N), –54.73 E 4, or DD(N), 54.73 E 4 S 10
   In this case the sign bit is set to one (negative) and the flag bits are zero.

c. DD(N), –54.73 E 4 S 5
   The sign bit is one, since the number is negative, and flag bits T and V are one. U is zero.

d. DD(N), 1, 3E–5, –45. 7, 12 S 17
   This example illustrates the multiple entry feature. This single DD statement compiles four 64-bit floating point words and advances the location counter accordingly.

e. DD(N), 1/3, 472*351, 4–7*5/21 S 4
   Note: Sign byte entered in last D field.

f. DD(N), 27.9/31.4/12/14 E 5, 4+3*7/5*6
   The number produced in the first case is:

$$\frac{27.9}{31.4 \times 12 \times 14 \times 10^5}$$

   in the second: $4 + \frac{3 \times 7 \times 6}{5}$.

g. DD(N), 1/7 – 3/11 + 1.4321 E – 2, .12 + 1/144
   As an extra convenience, certain system symbols are defined by which constants involving irrational numbers can be entered. They are:

1.	$PI	π
2.	$E	e
3.	$M	$\log_{10}e$
4.	$N	$\log_e 2$
5.	$INF	∞ (infinity)

Thus, one can enter a number such as $4\pi \times 10^{-7}$ by writing:

DD(N), 4 * $PI * 1E – 7.

Unnormalized Floating Point

Format:

Name | (Fn)DD(U), ± xx···x.x···xE±yyySn X±n

or

DD(U), (Fn) ± xx···xx.x···xE±yyySnX±n,
(Fn)±xx··· etc.

The number is converted to binary with the correct number of binary fractional places as specified by the (Fn) entry mode, and a correct exponent is computed and entered. This exponent is overruled and replaced by that following the X if X is used (necessary only if, for some reason, the programmer desires an incorrect exponent). The entry mode (Fn) can come before the DD, in which case it applies to all D fields of the statement, or it may form the first element of a D field, in which case it overrules one given before the DD. Either the X or the S or both may be omitted or their order may be interchanged. Omitting S has the same effect here as in the normalized case. Omitting X simply allows the correct exponent to remain as computed. Leaving out the sign, decimal point, or E is permitted as in normalized numbers.

Examples:

a. DD(U), (F21) – 343. 7, (F10) 432
   Two numbers are compiled. In the first, 343 is converted as an integer and .7 is converted to a 21-bit fraction. They are joined and placed in the rightmost bits of the fraction portion of the floating point word, and the correct exponent (in this case 27) and sign are supplied. In the second D field, 432 is converted to a binary integer. Because ten fractional bits are specified, but no decimal fraction is written, the ten rightmost bits of the fraction field are set to zero and the number is entered with its rightmost bit in position 50.

b. (F15)DD(U), 767. 52, 767. 52 X–12 S11
   The (F15) applies to both D fields. In the second, the computed exponent is overruled by the specified one and the number is made negative by means of the specified sign byte.

c. (F15)DD(U), 767.52, (F20) 767. 52 S11 X–12, 398
This example is identical to example b except that in the second field the operation entry mode (F15) is overruled by a field entry mode (F20), and the order of S and X is interchanged, which makes no difference. (F15) still applies to 398, however.

If the entry mode is omitted, two cases arise:

1) If the number entered is an integer, (F0) is understood.
2) If the number entered is a decimal fraction, it is converted to an unnormalized floating point number.

Examples:

a. DD(U), 17, 17X–35
In the first case 17 is converted to binary and placed in the fraction with its rightmost bit in position 60 and an exponent of 48 supplied. In the second field the same thing is done except that the exponent is set to –35.

b. DD(U), 17. 5
In this example 17. 5 is converted to normalized floating binary and stored as such. However, instructions whose normalization bits depend on the symbol in the name field of this pseudo-operation will have them set to unnormalized.

Note: 17 E 5 is an integer and will be recognized as such.
17 E–5 is a decimal fraction and will be normalized.
17. 5 E 5 is an integer but will be treated as a fraction and normalized. Thus, a normalized integer can be assigned use mode "unnormalized."
An integer greater than 2^{48} is stored as a normalized number.

Binary Signed VFL

Formats:

(Fn)DD(B, FL, BS), ± xx· · · x. x· · · xE±yy Sn
DD(B, FL, BS), (Fn) ±xx· · · x. x· · · xE±yy Sn
(R)DD(B, FL, BS), ±xx· · · xx Sn
DD(B, FL, BS), (R) ±xx· · · xx Sn

A data definition of binary signed data may have either (Fn) or (R) entry modes, but not both at the same time. (Fn) implies that the data following it are written in a decimal radix, whereas (R) implies that the number following it is an integer. An integer subject to a radix entry mode must be written without the aid of E because E is not defined for a radix other than 10. A decimal fraction must have a controlling (Fn) entry mode. There is no obvious way to convert to a fixed point number without specifying the binary scaling. In the data description either the field length or byte size or both may be omitted. The implied field length in this case is 64; the implied byte size is 1. The sign byte need not be specified unless the programmer desires to have flag or zone bits different from zero. Note that the sign bit position changes for a byte size less than 4. To make a number negative, specify the sign byte as:

BS = 1, S1
BS = 2, S2
BS = 3, S4
BS = 4, S10

If a number has no entry mode at all, it must be a decimal integer, but may in this case be written with the aid of the E notation.

Examples:

a. (F7)DD(B, ,4), . 005E3S13, –17, 143. 2S11, (8) 77760, 777
Implied field length is 64. Octal specification in the fourth D field overrules (F7)written before DD, but (F7) still applies to 777.

b. (2)DD(B, 16, 8) 110101S377, (10) –972, 11101110S201
Binary entry, overruled in only the second D field.

c. (F12)DD(B, 24), 1. 324E3, –72. 1E–4, 3. 4E–4S1
Implied byte size is 1.

d. DD(B), 1489, –1272, 1491, (F13) –972.16, 13948S1, 12E5
Decimal integers, except where a field entry mode is written.

Binary Unsigned VFL

Formats:

(Fn)DD(BU, FL, BS), xx· · · x. x· · · xE±yy
DD(BU, FL, BS), (Fn) xx· · · x. x· · · xE±yy
(R)DD(BU, FL, BS), xx· · · xx
DD(BU, FL, BS), (R) xx· · · xx
(Az)DD(BU, FL, BS), alphabetic information to "z"
(IQSz)DD(BU, FL, BS), alphabetic information to "z"
(Pz)DD(BU, FL, BS), alphabetic information to "z"
(CCz)DD(BU, FL, BS), alphabetic information to "z"

Numerical entry is exactly the same as in binary signed data except that no sign byte is formed, and if the byte size is left out of the dds, it is set to 8. Any sign or sign byte (with S) written with mode BU is ignored. The alphabetic modes are permitted here; they are explained under "Entry Modes." Note that the alphabetic entry mode must precede the DD, that there can be only one D field per statement, and that

if the field length is omitted, it is set equal to 64. If the byte size is omitted in entry mode CC, BS= 12 is implied.

Examples:

a. (F13)DD(BU, 30), 17.2, 183, (8) 70707

b. (A*)DD(BU, 48, 6), GLORIOUS FRIDAY, THE 13TH.*
The mode and field length have no effect on the conversion and storage; they are used in compiling instructions that refer to the name of this statement. Field length 48 indicates that the programmer wants to process these characters in groups of 8.

c. (IQSS)DD(BU, 32, 8) DOG EAT DOG S

Decimal Signed VFL

Formats: (R)DD(D, FL, BS), ± xx···xxx Sn
DD(D, FL, BS), ±(R) xx···xx Sn
DD(D, FL, BS), ± xx···xxEyy Sn
(Fn) has no meaning for mode = D or DU.

The two decimal modes in DD and DDI statements represent the only cases in which STRAP-II converts numbers to an internal decimal radix. The radix entry mode indicates the radix in which the numbers are written on the card. Thus, it is possible to write an integer in binary or octal and have it converted to decimal for machine use. If no entry mode is given, decimal to decimal is implied. The E notation can be used to multiply an integer by positive powers of 10. If either the field length or byte size is omitted, the implied values are FL = 64, and BS = 4.

Examples:

a. DD(D), −9534812, +173E5, 18E10S13
Field length = 64; byte size = 4. A 4-bit sign byte is formed. Decimal-to-decimal conversion.

b. (2)DD(D, 20), 11101000110157
Byte size = 4. Binary-to-decimal conversion.

c. DD(D, , 8), 432E3
Field length = 64. Decimal-to-decimal conversion. Four binary zeros are inserted in the zone positions of each byte.

Decimal Unsigned VFL

Formats:
(R)DD(DU, FL, BS), xx···xx
DD(DU, FL, BS), (R) xx···xx
DD(DU, FL, BS), xx···xxxEyyy
(Az)DD(DU, FL, BS), alphabetic information to "z"
(IQSz)DD(DU, FL, BS), alphabetic information to "z"

The numerical conversion is just as in decimal signed mode except for the omission of the sign byte. Alphabetic conversion is exactly as in the binary unsigned mode, except that instructions referring to these data are compiled as decimal operations. For alphabetic entry, implied field length is equal to byte size.

Examples:

a. DD(DU), 8430051, (8) 77241, 82E10
Field length = 64; byte size = 4.
An octal-to-decimal conversion is inserted between two decimal-to-decimal conversions.

b. (IQS3)DD(DU, , 8), PUSH PANIC BUTTON 3
Field length = 8.

Summary of Rules for DD Statements

ENTRY MODE	APPROPRIATE USE MODES
Fn	U, B, BU
R	B, BU, D, DU, N, U
A	BU, DU, U
IQS	BU, DU, U
CC	BU, DU, U
P	BU, DU, U

Note: Use mode N should have no entry mode.

SPECIAL FIELD ENTRY	APPROPRIATE USE MODES
S	N, U, B, D
X	N, U

The floating decimal notation, using E to designate multiplication by powers of 10, is appropriate to all modes.

If the field length is omitted from the dds, it will be assigned a value of 64. The maximum permissible field length for a DD statement is 64.

The parenthetical integer entry mode is appropriate in any DD statement, no matter what use mode has been written. The following examples illustrate the use of general parenthetical integer entry with DD:

a. DD(N), 572(.59)1, 347.89E12(.63, 2)1011
In the second case the sign byte is specified by means of (.n) entry.

b. DD(B), (F9) −35.7(.24) SAM + 4
The address SAM + 4 is placed in the first part of the 64-bit field, followed by the converted number −35.7.

c. (8)DD(BU), 4762(.10)707(10, .20)34
707 is written in octal, 34 in decimal.

d. DD(BU, 12(.2)7, 8), 787, 788
All numerals are in decimal. Binary 111 is ORed into the three high order bits of each 12-bit data field created.

Address Arithmetic

It is often convenient for a programmer to write an address expression composed of an arithmetic combination of two or more symbols, integers, bit addresses, etc. Relative addressing is a good example of the need for these type expressions. It has already been shown that the appearance of a $ in an address field has the meaning "the location of this very instruction." If one wishes to refer to the location exactly two full words beyond the location of the instruction containing the $, it may be preferable to write the address expression

$ + 2. 0

rather than to assign a programming symbol to this location and address the location by symbolic name. In another instance, a table is known to begin at symbolic location DATA and to be 20 full words in length. Clearly the full word immediately following the last word in the table can be addressed by the expression DATA+ 20. 0.

Many other situations can be imagined where address arithmetic would be desirable. STRAP offers generous provisions for the performance of address arithmetic. Virtually any mixture of STRAP bit addresses, integers, programmer symbols and system symbols can be combined by addition, subtraction, multiplication and division to form a single 24-bit standard binary bit address. From this point on the truncation (if necessary) and insertion of the bit address into the appropriate address field is completely standard.

Symbols for addition, subtraction, multiplication and division are the standard FORTRAN characters, that is + , −, * and / respectively. Addition and subtraction are the most common arithmetic operations and, when like quantities are involved, the procedure is completely straightforward. When two STRAP bit addresses are to be added together, the points are lined up and the two quantities are added. If two integers are to be added, the units positions are lined up before the addition is performed. In either of these cases, subtraction is analogous to addition.

Thus, the address expression in this instruction

L(BU), 8. +64. 0+12. 3

will be treated

```
   8. 0
  64. 0
+ 12. 3
  ------
  84. 3 = actual instruction address
```

In the case of integer expressions, such as

LI(BU, 18, 8), 8+2+13+1

the addition takes place

```
   8
   2
  13
+  1
  --
  24 = actual instruction address
```

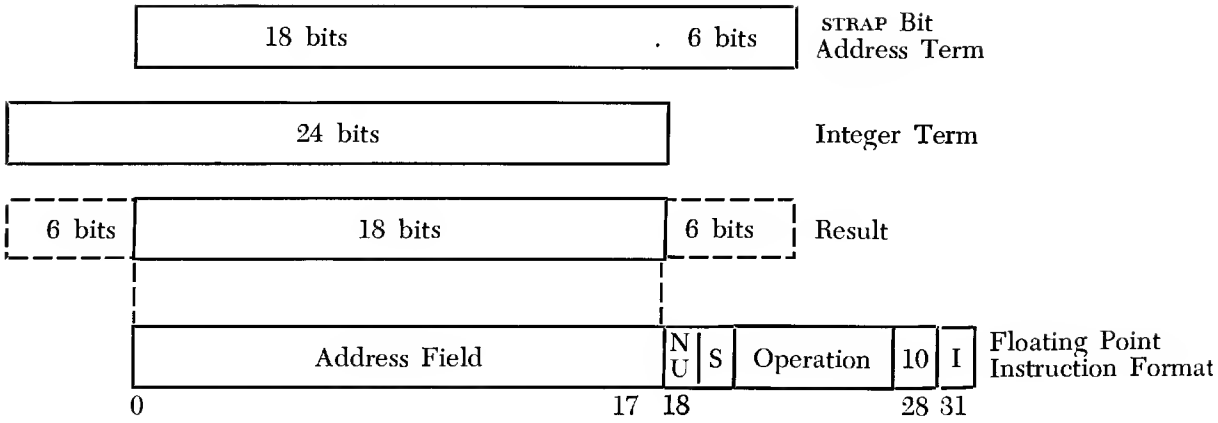
The sequence of steps that STRAP executes to perform addition and subtraction of like quantities in an address field is:

1. Convert each quantity to a 24-bit binary integer.
2. The quantities are aligned with respect to each other.
3. The numbers are assumed to be signed. Addition is algebraic.
4. The result is complemented if necessary. (Address fields are unsigned.) If the field is signed, such as an XW or VF, the sign bit is inserted in the correct bit and no complementation occurs.
5. The result is truncated, if necessary, to fit the particular address field.
6. The result is inserted into the correct position in the instruction.

When unlike quantities are added or subtracted, the sequence executed by STRAP is the same with the exception of a slight modification in Step 2. If integers and bit addresses are mixed, a certain amount of shifting, determined by the environment, must be performed before the addition takes place. For example, in the floating point instruction

+(N), 64. 0 + 20

the address field is 18 bits in length. The rule for positioning bit addresses is clear– the point must always line up between the 18th and 19th bits in the address field. Earlier it was explained that an integer is right justified in a field; here the units position falls in the 18th bit. Thus the two numbers are aligned



Up to this point, discussion has been limited to address fields. In reality, all previous statements apply to any field where arithmetic is permitted, that is any programmer symbolized field. Three restrictions must be observed.

1. No arithmetic may appear in the operation code part of the operation field, the mode subfield of the data description or any entry mode. All of these fields are reserved for designations whose meanings to STRAP are absolute and may not be symbolized.
2. No arithmetic may appear in the name field, which is reserved entirely for the definition of symbols. Only one symbol per statement is allowed.
3. The "I" or "K" fields (see Expression of Machine Instructions) must contain at least *one* STRAP bit address term.

The diagrams below illustrate the complete set of rules for shifting and truncation that cover addition and subtraction of unlike quantities in all 7030 instruction fields where arithmetic is permitted. The two basic precepts involved are:

1. Where a bit address has some meaning, the point is positioned between the 18th and 19th bits of the field. If a bit address has no meaning, the entire 24-bit quantity is treated as an integer and right justified in the field. Index fields are an exception.
2. An integer is always treated as an integer in the environment that is the size of the particular field. The integer is right justified so that its units position is aligned with the units position of the field.

Although the diagrams show the final sum truncated to the appropriate length, the bits are not actually discarded unless they fall outside the address field of the instruction. Some operations do not use all of the space available in their address fields (transmit, input-output select), and in these cases bits may be placed in the unused portions.

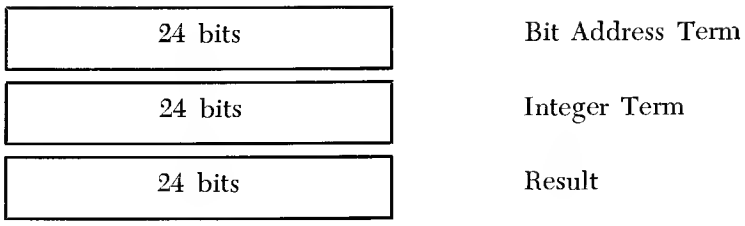
An error indication is given if non-zero bits are discarded when truncation occurs, except in the case of index fields where a "1" bit in the fifth position from the right (in the "16" position) is discarded without error indication.

Truncation occurs for particular fields in the following manner:

1. A_{24} Bit Address

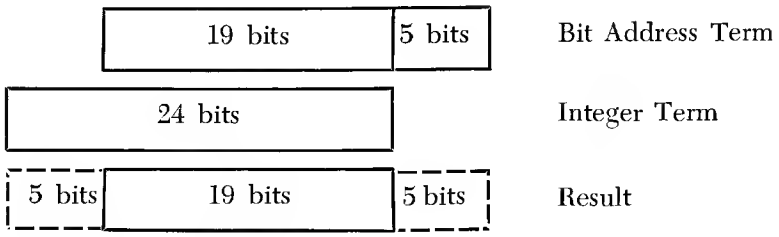
   Rule: No truncation

   Note: An integer in a 24-bit field counts bits



2. A_{19} Half-Word Address

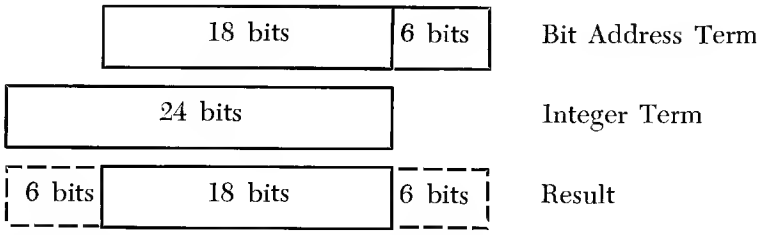
   Rule: Leftmost 5 bits and rightmost 5 bits are truncated from sum



   Note: An integer in a 19-bit field counts half-words

3. A_{18} Full-Word Address

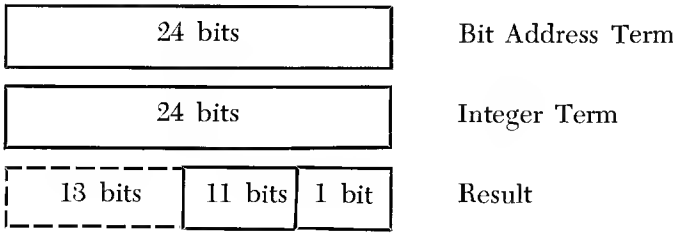
   Rule: Leftmost 6 and rightmost 6 bits are truncated from the sum



   Note: An integer in an 18-bit field counts full words or unit address, control operation, control word address, and so on, in right I-O address.

4. $A_{11\pm}$ Signed 11-Bit Address

   Rule: Leftmost 13 bits are truncated from the sum. Rightmost 11 bits plus sign are placed in leftmost 12 bits of address field of shift and Add Immediate to Exponent instructions

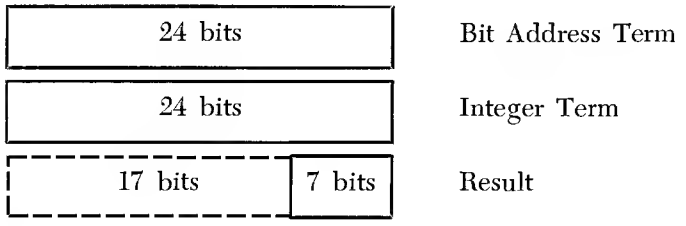


   Note: Integer counts number of bits in shift or number of bits to be added to exponent of floating point word.

5. OF_7 Offset

   Rule: Leftmost 17 bits of sum are truncated

   Note: Integers count number of bits of offset

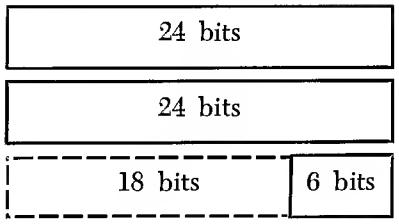


   Bit address 1.32 = .96 = integer 96

6. FL_6 Field Length

Rule: Leftmost 18 bits of sum are truncated

Note: Integers count length of field in bits



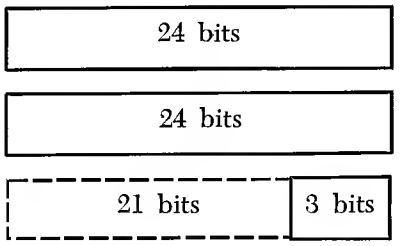
Bit address 1.0 = .64 = 0 not error marked

7. BS_3 Byte Size

Rule: Leftmost 21 bits of sum are truncated

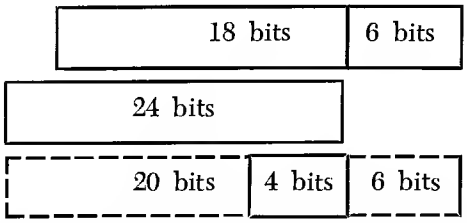
Note: Integers count byte size in bits

.8 = 8 = 0 not error marked



8. I, J 4-Bit Index Fields

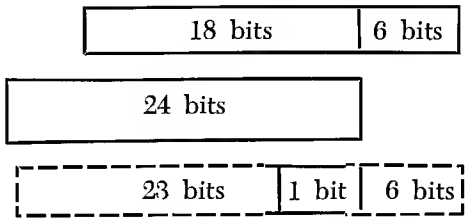
Rule: Leftmost 20 bits and rightmost 6 bits of sum are truncated



Note: Integers represent index register number. A "1" in the bit position immediately to the left of the final sum field is discarded with no error indication.

9. K Single Bit Index Field

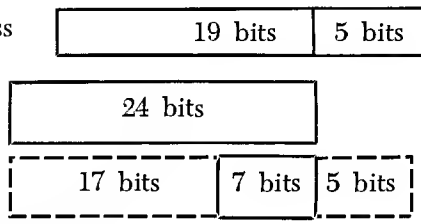
Rule: Leftmost 23 bits and rightmost 6 bits of sum are truncated



Note: Integers specify either index register 0 or index register 1. A "1" in the bit position that corresponds to "16" in the sum is discarded with no error indication.

10. A_7 I-O Left Effective Address

Rule: Leftmost 17 and rightmost 5 bits are truncated from sum



Note: Integers specify channel address

One exceptional condition must be noted here. This is the case of immediate operation address fields. In this instance, the treatment of a mixed expression consisting of both integers and bit addresses differs from the general rules above. The treatment of integers is straightforward and the result is left justified before insertion in the field. (See DD1). If two or more bit address terms are being combined, the arithmetic is as usual but no left justification is done. The field length in the dds is ignored and the point is lined up between the 18th and 19th bits as in any other field. However, when integer and bit address terms are to be combined, all terms are considered to be bit addresses; they are aligned accordingly and the result is inserted as a bit address. The following immediate operation

```
LI(BU, 24, 8), 2+2.2+6
```

is treated by STRAP as if it had been written

```
LI(BU, 24, 8), .2+2.2+.6
```

Programmer symbols, defined elsewhere in the code as integers or STRAP bit addresses, may participate in the address arithmetic and no restrictions other than those already outlined need be observed. System symbols defined as bit addresses may also be used. Therefore

```
ANKLEBONE SYN, 20.2
 FOOTBONE SYN, 1
          L(BU),FOOTBONE + ANKLEBONE -2 + 888.08
```

is perfectly legal, while

```
CIRCLE ST(BU), CIRCLE -$PI
```

is not.

There is no limitation on the number of terms that may appear in an arithmetic expression. Continuation card(s) must be used if the expression exceeds the space available on the symbolic card.

Arithmetic expressions involving multiplication and division are handled somewhat differently by STRAP. Here the assembly program recognizes that certain combinations (two or more integers or integers and bit addresses) can have meaningful results while multiplying or dividing two or more bit addresses has little meaning so that, although such operations are not prohibited, arbitrary rules are imposed on the arithmetic.

The basic precept in multiplication and division is that both bit address terms and integer terms are treated as 24-bit integers and the point is forgotten in the bit address once the conversion to binary is accomplished. This means that the address expression

```
2.0 * 2
```

is the same as writing

```
128 * 2
```

No shifting is done.

The two numbers are simply assumed to be integers, are aligned with respect to each other and are multiplied or divided on this basis. The result is also treated as an integer, that is, it is right justified in the field in which it is being inserted. If the field is smaller than 24 bits in length, all truncation occurs on the left.

The sequence that STRAP follows then to multiply or divide an address expression that is a mixture of bit addresses and integers is

1. Convert all terms to 24-bit binary integers.
2. Assume all terms are signed integers. Multiply or divide as requested.
3. The result is complemented if necessary
4. The result is truncated on the left if necessary to fit the particular field.
5. The result is inserted in the field as an integer, i.e. it is right justified in the field.

An illustration of multiplication in an address field will point out how three different expressions using the same numbers will produce three very different results. In the first case

```
CM1010(BU), 2 * 2($X7)
```

multiplication of two integers proceeds as would be expected and the arithmetic

```
   2
 × 2
 ---
   4
```

If, however, the instruction had been written

```
CM1010(BU), 2.0 * 2
```

the multiplication would now be performed in this manner

```
 128
 ×  2
 ----
 256
```

In still a third case, this address expression

```
CM1010(BU), 2.0 * 2.0
```

which is multiplied by STRAP as

```
 128
 × 128
 -----
16384
```

The STRAP bit address, when converted to 24-bit binary integer form, is specifying an integral number of bits. The 24-bit representation of integer terms is also a number of bits. The results, therefore, are also treated as an integral number of bits. In case one, the answer is 4 bits as one would expect when multiplying two bits by two bits. In case two, the answer is 256 bits or four words, also to be anticipated when multi-

plying two words by two. However, case three presents a multiplication of two bit addresses wherein the results can only be arbitrarily defined, here 16384 bits or 256 full words.

The result of multiplication or division can be forced to be interpreted by STRAP as a bit address. If the expression is enclosed in parentheses and followed by a period, the result will be treated like a standard binary bit address, that is, it will be appended by six zeros and inserted in the address field with the period lined up between the 18th and 19th bits. Truncation, if required, will be performed in the manner specified for bit addresses. To illustrate, the address expression in this instruction

M+(BU), 200 * 50

yields a result of 10000 which, when inserted in this address field as an integer, would count bits. If the expression had been written

M+(BU), (200 * 50).

the result 10000 would now be treated as a bit address, or 10000.0, which would count full words.

Two other alternatives are possible. The instruction could be written

M+(BU), 200.0 * 50

where the result is 640,000 which is treated as an integer and inserted in the field when compiled to yield an integral bit count. Again, by use of the special notation

M+(BU), (200.0 * 50.0).

bit address characteristics are attached to the integer result, yielding an address of 640,000.0.

It should be pointed out that an expression comprised of all four types of arithmetic operations is perfectly legal. The instruction

SRD(BU), 200 + 70.0 – 600 * 2 / 4

is perfectly legal. In an expression of this type, STRAP performs the arithmetic operations in the following order—multiplication, division, addition and subtraction. The treatment of each term is strictly in accordance with the rules described above.

Additional Pseudo Operations

There are several STRAP pseudo operations that perform rather specialized functions. These fall into three categories:

Output Listing Pseudo Operations

1. PRNS—Print Single Spaced

PRNS

This pseudo operation causes the assembly listing to be printed with single spacing. Double spacing is the normal printing mode, and is the mode in effect for every assembly except those in which PRNS is specifically written.

2. PRND—Print Double Spaced

PRND

This pseudo operation restores printing to the normal double spacing mode after the use of a PRNS. At the conclusion of each assembly, the mode is automatically reset to double space, so that PRNS need only be used if it is desired to change mode from single to double space in the middle of one assembly.

3. NOPRNT—No Printing

NOPRNT

This pseudo operation stops printing of the output listing until any other printing pseudo operation is encountered in the program, at which time printing is resumed.

4. SPNUS—Suppress Printing of Unused Symbols

SPNUS

This pseudo operation suppresses printing of the list of unused symbols that appears at the beginning of the output listing (see STRAP-II Output Listing). The list is suppressed for the compilation of the entire program in which the SPNUS appears. Printing of the list is not restored until the beginning of the next assembly.

Output Punching Pseudo Operations

1. PUNFUL—Punch Full Cards

PUNFUL

Full cards (80 columns of column binary information) are punched without checksum, first word address, identification, and so on.

2. PUNNOR—Punch Normally

PUNNOR

This pseudo operation restores normal punching (72 columns) after the use of a PUNFUL.

3. PUNORG—Punch Origin

PUNORG

This pseudo operation causes an origin to be punched in every binary card in the output deck, thus making every binary card produced by STRAP II an origin card.

4. NOPUN—No Punch

NOPUN

Punching of the binary output deck by STRAP II can be halted by the use of the NOPUN pseudo operation. Punching remains suppressed until a PUNNOR or PUNFUL pseudo operation is encountered.

5. PUNSYM—Punch Cards For Symbols

PUNSYM, A,A',A'',...A_n

The A_i are any legal programmer symbols that are used elsewhere in the program. After the entire binary deck has been punched out, one card in the following format will be punched out for each A_i specified.

Card Column	Contents
1	
2-9	Programmer Symbol
10-12	SYN
13-21	dds
22-25	,(8)
26-28	blank
29	sign
30-38	Bit address (xxxxxx.xx)
39-41	blank
42-44	(8)
45	integer sign
46-53	integer
54-55	blank
56-60	Index value ($xx)
61-72	Array dimensions
73-80	ID specified by the latest PUNID

The SYN cards thus produced permit reassembly of a portion of a program that refers to symbols defined in another portion not being reassembled. The SYN cards will be put in the symbol table at reassembly time, and the symbols involved are thereby legally defined.

The format of the card produced by PUNSYM allows for symbols that are defined as bit addresses, integers or arrays. (See Array Specification Using DR below.) When a symbol has been defined as an integer somewhere in the program, PUNSYM yields a card that has the integer definition in columns 44-55, and the fields reserved for a bit address definition or an array definition are left blank. Note the presence of the radix specifier which denotes that bit address and integer definitions are always punched in octal.

If the symbol is too long to fit in the name field of one card, STRAP will automatically supply a continuation card or cards. Similarly, if the array definition is too long to fit on one card, a continuation card is supplied and the definition is continued beginning in column 10.

6. PUNALL—Punch All

PUNALL

This pseudo operation causes STRAP to punch a SYN card for every symbol used in the program.

Miscellaneous Pseudo Operations

1. TAIL—Tail

TAIL, ANYSYMBOL

Difficulty with multiply-defined symbols can arise when two programs, written by different people at different locations, are assembled together. By appending a unique programmer symbol as a tail to every symbol in his program, a programmer can be assured that each of his symbols will be uniquely defined, regardless of what other programs are assembled with his program.

The pseudo operation TAIL appends the symbol that appears in its address field as a tail to every symbol in the statements that follow the tail statement until another tail statement, or an untail statement, is given.

A tail symbol can be any legal programmer symbol; it may be composed of as many as 128 alphanumeric characters, the first of which is specifically alphabetic. When tailing is used, the last two characters of the basic symbol are used for a special character that indicates tailing is being used and a character to represent the tail symbol. Therefore, a programmer symbol of more than 126 characters cannot be tailed. As many as 256 distinct tail symbols can be used within any one program.

STRAP-II permits up to ten levels of tailing; that is, as many as ten different tail symbols may be appended to each programmer symbol within a block of code. When only one level of tailing is used, two characters must be subtracted from the maximum size of a programmer symbol to be tailed. In multi-level tailing, an additional character must be subtracted for each additional level of tailing. If n=the number of levels of tailing, n+1 characters must be subtracted from the maximum size programmer symbol. Thus, if 6 levels of tailing are to be used, the maximum size programmer symbol that may appear in that tailed block is 121 characters in length; when ten level tailing is specified, the longest programmer symbol may be 117 characters in length.

To facilitate multi-level tailing, a sub-field is added to the basic tail statement format, as in

TAIL, (n)DOG

where n refers to the level of tailing to which the given tail symbol is to be assigned. If the level is not

specified, the first level is assumed. Thus,

TAIL, (1)DOG

can also be written

TAIL, DOG

Omission of the parentheses as in

TAIL, 1DOG

will result in an illegal tail symbol and invalidate the statement.

The tail will continue to be added to every programmer symbol encountered at the level specified until an untail statement or a tail statement that specifies the same level is found. An untail statement will untail all levels up to and including the level specified in the address field. The statement

TAIL, (6)DOG

specifies DOG as a sixth level tail, and

UNTAIL, (6)

untails the first six levels. Note that

UNTAIL, (1)

is equivalent to

TAIL, (1)

but

UNTAIL, (2)

is not equivalent to

TAIL, (2)

since UNTAIL, (2) will untail the first and second level, while TAIL, (2) tails the second level only with a blank, or effectively untails it. Clearly then, if it is desired to untail one level when multi-level tailing is being done, the best method is a tail statement that specifies the level but has a blank tail symbol field, as in

TAIL, (6)

The normal reference may be made from one symbol to another within the same tailed block. However, when reference is made from a block tailed by DOG, for example, to a possible multiply defined symbol BOB in another block tailed by CAT, the 7030 statement should read

+(N), BOB$CAT

If the symbol BOB has been tailed at several levels, they must all be mentioned:

+(N), BOBCATTAYLE

If reference is made from a tailed block to a possible multiply defined symbol in an untailed block, only the $ is required after the symbol, as in

+(N), BOB$

The $ alone (actually $ followed by a blank) tells STRAP that the reference is to the untailed symbol BOB, not the BOB defined in the tailed block.

2. EXT—Extract

A EXT,(I, J, COUNT)STATEMENT

The Extract pseudo operation has the following meaning:

First, compile STATEMENT as if it were any legal 7030 instruction or pseudo operation that produces binary output. Then extract from this statement the subfield that is equal in length to the number of bits specified by COUNT and begins at bit I of that statement and ends at bit J. The extracted subfield is then actually compiled in the position in the code where the EXT occurs.

Any symbol A appearing in the Name field is assigned a data description BU, a field length equal to COUNT (or J–I+1), and a byte size of 8, and is attached to the subfield compiled.

Any 2 of the 3 parameters I, J, and COUNT are sufficient to adequately describe the subfield to be extracted. All 3 can be written if the programmer so desires, but if less than 3 are written, the usual right to left drop out order rules, as in the dds. Therefore, the permissable alternatives are:

(I, J, COUNT)
(I, J)
(I, , COUNT)
(, J, COUNT)

The terms I, J, and COUNT may contain any number of symbolic integers. A bit address is improper, however, and will be treated as a 24-bit binary integer.

If EXT is used to specify the extraction of anything beyond the range of the single statement that follows it up to 64 zeros will be added.

Example: EXT(18, 47)+(B, 18, 7), 73.16

First the full word instruction +(B, 18, 7), 73.16 is formed. Then bits 18 through 47 (the first bit in the instruction is numbered zero according to 7030 custom) are extracted and placed in the program being compiled. The dds (BU, 30, 8) is formed. The location counter is advanced 30 bits.

3. CNOP—Conditional No Operation

A CNOP

The pseudo operation CNOP is used to insure that the instruction or data immediately following the CNOP will be assigned a full-word address by STRAP II.

When a CNOP is encountered, the location counter is immediately rounded up to the nearest half-word address if it is not already at a half-word address. Then STRAP examines the location counter. If it now stands at a full-word address, the CNOP is ignored.

If, however, the location counter is set to a half-word address, the 7030 instruction NOP is compiled. This has the effect of advancing the location counter 32 bits or one-half word to the next full-word address.

Any symbol A appearing in the Name field is assigned a full-word address when the CNOP is ignored, or a half-word address when a NOP is compiled.

In the following example:

```
        SLC, 100.32
CASE1   CNOP
        L(BU, 24, 8), ASSIST
CASE2   CNOP
        +(N), FLOATINGONE
```

the appearance of the first CNOP causes a 7030 NOP instruction to be compiled at location 100.32. The Load instruction is compiled at 101.0. The symbol CASE1 is assigned the value 100.32. When the second CNOP is encountered, the location counter stands at 102.0. The CNOP is then ignored, the floating point Add instruction is compiled at storage location 102.0 and the programmer symbol CASE2 is assigned the value 102.0. Thus CASE2 becomes the symbolic location of the floating point instruction.

4. TLB—Terminate Loading and Branch

```
TLB, Y
```

The pseudo operation TLB is similar to the END statement with one major distinction: it does not stop the assembly process. Therefore, TLB may be assembled at any point in the symbolic deck where a transition card is desired. The branch card thus produced will interrupt the loader when encountered in a binary deck and transfer control to the instruction at location Y. The remainder of the program must be loaded under program control.

5. SLCR—Set Location Counter Relative

```
SLCR, Y
```

SLCR resets the STRAP location counter to the address Y in much the same fashion as SLC. However, SLCR also stops binary punching, so that locations of statements following SLCR are assigned relative to the location specified in SLCR but none of the statements appear in the binary output. This effect is the same as if all symbols in the name field of the statements that follow the SLCR were defined by SYN statements, and the convenience for the programmer is more desirable.

In the most common usage,

```
SLCR, 0
```

will reset the location counter to 0, and all symbols following are assigned locations relative to 0. A useful application of this use of SLCR might occur in the definition of table formats. In the following sequence

```
        SLCR, 0
PRICE   DD(BU, 24, 8), 0
QUANTITY DD(BU, 6, 8), 0
ONHAND  DD(BU, 10, 8), 0
```

the evaluation of the symbols will be

```
PRICE = 0.0
QUANTITY = 0.24
ONHAND = 0.30
```

If the table in question begins at location 2000.0, and this address is placed in the value field of index register 6, the relative addressing of items in the table can be accomplished in simple fashion as shown in these instructions:

```
L, QUANTITY($6)
*, PRICE($6)
```

These instructions would be compiled by STRAP as

```
L, .24($6)
*, 0.0($6)
```

One advantage of this method is the ease with which the dds of one of the statements can be changed without requiring changes in any of the others. The definitions can be reordered also with no other changes in the statements required and all address assignments are recomputed by STRAP relative to the SLCR address.

SLCR is allowed to set the location counter to an address below 41_8 without causing an error message to be printed. This is not the case if SLC had been used. The locations subsequently assigned will often be below 41_8 as well, but they are usually indexed to produce addresses above the first 32 storage locations. In many ways SLCR is equivalent to SLC followed by a NOPUN. An SLC must be issued to restore binary punching of the output deck.

6. SEM—Suppress Error Messages

```
SEM, 1, 2, 3, ...
```

The pseudo operation code SEM, followed by a blank address field, causes all error messages detected in statement that follow the SEM statement to be suppressed on the output listing. Any particular message or group of messages may be suppressed by writing the numbers identifying the messages in the address field, separated by commas. Thus,

```
SEM, 8, 2
```

suppresses the printing of error messages 2 and 8 only.

7. REM—Resume Error Messages

```
REM, 1, 2, 3, ...
```

An REM restores normal error message printing on

the listing after an SEM has been used. The ability to specify individual messages or all messages at once is also available with REM. Thus, following the statement

SEM, 9, 16, 18

the pscudo operation

REM, 16

restores normal error printing to message 16, while messages 9 and 18 remain suppressed.

8. LINK—Link

LINK

The LINK pseudo operation provides the programmer with a shorthand notation for an entry or linkage into a subroutine. At the point in the code where the LINK is encountered, STRAP substitutes the 7030 operation

LVI, $15, $+2

which follows the custom of using index register 15 to store the instruction counter value of the return instruction and has become the standard entry mechanism.

9. DR also provides a convenient method of defining multidimensional arrays of data and of addressing individual elements of arrays so defined. All indexing required for the manipulation of the array must be handled by the programmer.

The statement:

A DR(dds), (L, L', L'',...,L^r)

reserves space for an L × L' × L''×...L^r array of data fields. The location counter is skipped forward a number of bits equal to the field length (specified in the dds) multiplied by the product of the dimensions of the array. (If the dds specifies the floating point mode, the correct number of full-words is reserved, beginning at a full-word boundary.)

Any symbol A appearing in the name field is attached to the first element of the array, and the dds is attached to the symbol in the normal fashion. Thus, in an instruction, a specific element of the array may be addressed by writing:

A (q, q', q'',...,q^r)

Note that the first element of the array has the address:

A (0,0,0,...,0)

and the last element is located at:

A (L–1, L'–1, L''–1,...,L^r–1)

The address of an arbitrary element in the array may be computed by means of the formula:

Address of A(q,q',q'',...,q^r) = Address of A(0, 0,0,...0) +FL×(q+q'L+q''LL'+q'''LL'L''+ ...)

where FL is the field length of any element in the array. An array address computed in this manner may be used in any programmer symbolized field not in parentheses, except a general parenthetical integer entry. The dimension of a DR statement must be evaluated by the end of pass 1. Therefore, they may be defined by a chain of SYN's ending in an integer.

L, L', L'', etc., must be integers in symbolic or numeric form. Referring to "Address Field" to apply index register I to the second element of a one dimensional array A, write:

A(1)(I)

where I must be a bit address.

SYN must be used to define a symbol as an interior element of a multidimensional array and have the dimensional addressing properties carried along. For example:

Name	Statement
A	DR(N), (10,20)
B	SYN, A(5,5)

In the above example, the rectangular array goes from A(0, 0) to A(9, 19); B goes from B(–5, –5) to B(4, 14); A and B use identical storage. Thus, A(0, 0) –B (–5, –5); A(1, 0) –B(–4, –5); A(1, 1) –B(–4, –4); etc.

STRAP-II APPENDICES

APPENDIX A

STRAP-II MNEMONICS

Assigned STRAP-II mnemonics, including both operation codes and system symbols, are listed on the following pages. The numbers in the Footnote column designate notes that follow the listing. These footnotes, in general, identify a particular class of operations that may be expanded in a standard way to produce other operations. Where footnotes specify how particular modified operation mnemonics may be constructed, these mnemonics do not appear explicitly in the listings.

The following abbreviations, used in the Type column, identify the symbolic instruction type.

V	VFL
F	Floating Point
$	System Symbol
I	Index
C	Count and Branch
M	Branches and Miscellaneous
B	Branch on Bit
T	Transmits
E	I-O Select or Control Word

Type	Mnemonic	Footnote	Name	Word No.	Bit Address
$	AD	2	Address Invalid	11	16
$	AE	2	Accumulator Equal	11	61
$	AH	2	Accumulator High	11	62
$	AL	2	Accumulator Low	11	60
$	AOC	1	All Ones Count	7	44-50
$	BC	1	Boundary Control	3	57
$	BTR	2	Binary Transit	11	39
$	CA	1	Channel Address	5	12-18
$	CBJ	2	Channel Busy Reject	11	8
$	CNSL	1	Console		
$	CPUS	1	CPU Signal	11	5
$	CPU	2	Other CPU	6	0-18
$	CS	2	Channel Signal	11	13
$	DF	2	Data Fetch	11	20
$	DISK	1	Disk		
$	DS	2	Data Store	11	19
$	DTR	2	Decimal Transit	11	40
$	E	12	e		
$	EE	2	End Exception	11	11
$	EK	2	Exchange Control Check	11	3
$	EKJ	2	Exchange Check Reject	11	6
$	EOP	2	End of Operation	11	12
$	EPGK	2	Exchange Program Check	11	9
$	EXE	2	Execute Exception	11	18
$	FT	1	Factor	14	0-63
$	IA	1	Interruption Address	2	0-17
$	IF	2	Instruction Fetch	11	21
$	IK	2	Instruction Check	11	1
$	IJ	2	Instruction Reject	11	2
$	IND	1	Indicators	11	0-63
$	IQS	1	Inquiry Station		
$	IR	2	Imaginary Root	11	25
$	IT	1	Interval Timer	1	0-18
$	L	1	Left Half of Accumulator	8	0-63
$	LB	1	Lower Boundary	3	32-49
$	LC	2	Lost Carry	11	22
$	LS	2	Lost Significance	11	26
$	LZC	1	Left Zeroes Count	7	17-23
$	M	12	$\text{Log}_{10}e$		
$	MASK	1	Mask	12	21-49
$	MB	1	Maintenance Bits	4	0-63
$	MK	2	Machine Check	11	0
$	MOP	2	To-Memory Operation	11	55
$	N	12	$\text{Log}_e 2$		
$	NM	2	Noisy Mode	11	63
$	OP	2	Operation Invalid	11	15
$	PCH	1	Punch		
$	PF	2	Partial Field	11	23
$	PGO...PG 6	2	Program Indicators	11	41-47
$	PI	12	π		
$	PRT	1	Printer		
$	PSH	2	Preparatory Shift Greater Than 48	11	27
$	R	1	Right Half of Accumulator	9	0-63
$	RDR	1	Reader		
$	RGZ	2	Result Greater Than Zero	11	58
$	RLZ	2	Result Less Than Zero	11	56
$	RM	1	Remainder	13	0-63
$	RN	2	Result Negative	11	59
$	RU	2	Remainder Underflow	11	34
$	RZ	2	Result Zero	11	57
$	SB	1	Sign Byte	10	0-7
$	TC	1	Time Clock	1	28-63
$	TCI...TCK		Tape Chanels 1...K		
$	TF	2	T Flag	11	35
$	TR	1	Transit	15	0-63
$	TS	2	Time Signal	11	4
$	TX	1	Tape X (X is a numerical designation)		
$	UB	1	Upper Boundary	3	0-17
$	UF	2	U Flag	11	36
$	UK	2	Unit Check	11	10
$	UNRJ	2	Unit Not Ready Reject	11	7
$	USA	2	Unended Sequence of Addresses	11	17
$	VF	2	V Flag	11	37
$	X0	1	Index Zero	16	0-63
$	X1	1	Index One	17	0-63
$	X2	1	Index Two	18	0-63
$	X3	1	Index Three	19	0-63
$	X4	1	Index Four	20	0-63
$	X5	1	Index Five	21	0-63
$	X6	1	Index Six	22	0-63
$	X7	1	Index Seven	23	0-63
$	X8	1	Index Eight	24	0-63
$	X9	1	Index Nine	25	0-63
$	X10	1	Index Ten	26	0-63
$	X11	1	Index Eleven	27	0-63
$	X12	1	Index Twelve	28	0-63
$	X13	1	Index Thirteen	29	0-63
$	X14	1	Index Fourteen	30	0-63
$	X15	1	Index Fifteen	31	0-63
$	XCZ	2	Index Count Zero	11	48
$	XE	2	Index Equal	11	53
$	XF	2	Index Flag	11	38
$	XH	2	Index High	11	54
$	XL	2	Index Low	11	52
$	ZM	2	Zero Multiply	11	33
$	XPFP	2	Exponent Flag Positive	11	28
$	XPH	2	Exponent Range High	11	30
$	XPL	2	Exponent Range Low	11	31
$	XPO	2	Exponent Overflow	11	29
$	XPU	2	Exponent Underflow	11	32
$	XVGZ	2	Index Value Greater Than Zero	11	51
$	XVLZ	2	Index Value Less Than Zero	11	49
$	XVZ	2	Index Value Zero	11	50
$	Z	1	Word Number Zero	0	0-63
$	ZD	2	Zero Divisor	11	24

ALPHABETIC LIST OF OPERATIONS

Type	Mnemonic	Footnote	Name
V	+	3	Add
F	+	6	Add
V	+MG	3	Add to Magnitude
F	+MG	6	Add to Magnitude
V	–	3	Subtract
F	–	6	Subtract
V	–MG	3	Subtract from Magnitude
F	–MG	6	Subtract from Magnitude
V	*	4	Multiply
F	*	7	Multiply
V	*+		Multiply and Add
F	*+		Multiply and Add
F	*A +		Multiply Absolute and Add
V	*I +		Multiply Immediate and Add
V	*N +		Multiply Negative and Add
F	*N +		Multiply Negative and Add
F	*NA +		Multiply Negative Absolute and Add
V	*NI +		Multiply Negative Immediate and Add
V	/	4	Divide
F	/	7	Divide
M	B		Branch
B	BB		Branch on Bit
B	BB1		Branch on Bit and Set to One
B	BBN		Branch on Bit and Negate
B	BBZ		Branch on Bit and Zero
M	BD		Branch Disabled
M	BE		Branch Enabled
M	BEW		Branch Enabled and Wait
M	BR		Branch Relative
B	BZB		Branch on Zero Bit
B	BZB1		Branch on Zero Bit and Set to One
B	BZBN		Branch on Zero Bit and Negate
B	BZBZ		Branch on Zero Bit and Zero
V	C	10	Connect
I	C + I		Add Immediate to Count
I	C – I		Subtract Immediate from Count
C	CB	8	Count and Branch
C	CBR	8	Count, Branch, and Refill
C	CBZ	8	Count and Branch on Zero Count
C	CBZR	8	Count, Branch on Zero Count, and Refill
E	CCW		Copy Control Word
V	CM	10	Connect to Memory
V	CT	10	Connect for Test
E	CTL		Control
V	CV	5	Convert
F	D +	6	Add Double
F	D + MC	6	Add Double to Magnitude
F	D –	6	Subtract Double
F	D – MG	6	Subtract Double from Magnitude
V	DCV	5	Convert Double
F	DL	7	Load Double
F	DLWF	7	Load Double with Flag
F	D*	7	Multiply Double
F	D/	7	Divide Double
F	E +	6	Add to Exponent
F	E + AI		Add Absolute Immediate to Exponent
F	E + I		Add Immediate to Exponent
F	E –	6	Subtract from Exponent
F	E – AI		Subtract Absolute Immediate from Exponent
F	E – I		Subtract Immediate from Exponent
M	EX		Execute
M	EXIC		Execute Indirect and Count
F	F +	6	Add to Fraction
F	F –	6	Subtract from Fraction
V	K	4	Compare
F	K	7	Compare
I	KC		Compare Count
I	KCI		Compare Count Immediate
V	KE	4	Compare If Equal
V	KF	4	Compare Field
V	KFE	4	Compare Field If Equal
V	KFR	4	Compare Field for Range
E	KLN		Check Light On
F	KMG	7	Compare Magnitude
F	KMGR	7	Compare Magnitude for Range
V	KR	4	Compare for Range
F	KR	7	Compare for Range
I	KV		Compare Value
I	KVI		Compare Value Immediate
I	KVNI		Compare Value Negative Immediate
V	L	4	Load
F	L	7	Load
I	LC		Load Count
I	LCI		Load Count Immediate
V	LCV	4	Load Converted
V	LF		Load Field
V	LFT	4	Load Factor
F	LFT	7	Load Factor
E	LOC		Locate (same as Select Unit)
I	LR		Load Refill
I	LRI		Load Refill Immediate
I	LV		Load Value
I	LVE		Load Value Effective
I	LVI		Load Value Immediate
I	LVNI		Load Value Negative Immediate
I	LVS		Load Value with Sum
I	LX		Load Index
V	LTRCV	4	Load Transit Converted
V	LTRS	4	Load Transit and Set
V	LWF	4	Load with Flag
F	LWF	7	Load with Flag
V	M +		Add to Memory
F	M +	6	Add to Memory
V	M + 1		Add One to Memory
F	M + A		Add to Absolute Memory
V	M + MG	3	Add Magnitude to Memory
F	M + MG	6	Add Magnitude to Memory
V	M –		Subtract from Memory
F	M –		Subtract from Memory
V	M – 1		Subtract One from Memory
F	M – A		Subtract from Absolute Memory
V	M – MG	3	Subtract Magnitude from Memory
F	M – MG	6	Subtract Magnitude from Memory
M	NOP		No Operation
M	R		Refill
M	RCZ		Refiill on Count Zero
E	RD		Read
E	REL		Release
E	REW		Rewind
I	RNX		Rename
F	R/		Reciprocal Divide
I	SC		Store Count
E	SEOP	11	Suppress End of Operation
V	SF		Store Field
F	SHF	7	Shift Fraction
F	SHFL		Shift Fraction Left (same as SHFA)
F	SHFR		Shift Fraction Right (same as SHFNA)
M	SIC		Store Instruction Counter If
F	SLO	7	Store Low Order
F	SNRT	6	Store Negative Root
I	SR		Store Refill
V	SRD	5	Store Rounded
F	SRD	7	Store Rounded
F	SRT	6	Store Root
V	ST	5	Store
F	ST	7	Store
E	SU		Select Unit (same as Locate)
I	SV		Store Value
I	SVA		Store Value in Address
T	SWAP		Swap
T	SWAPI		Swap Immediate
T	SWAPB		Swap Backward
T	SWAPBI		Swap Backward Immediate
I	SX		Store Index
T	T		Transmit
T	TI		Transmit Immediate
T	TB		Transmit Backward
T	TBI		Transmit Backward Immediate
I	V +		Add to Value
I	V + I	9	Add Immediate to Value
I	V + C		Add to Value and Count
I	V + CR		Add to Value, Count, and Refill
I	V + IC	9	Add Immediate to Value and Count

Type	Mnemonic	Footnote	Name
I	V + ICR	9	Add Immediate to Value, Count, and Refill
I	V – I	9	Subtract Immediate from Value
I	V – IC	9	Subtract Immediate from Value and Count
I	V – ICR	9	Subtract Immediate from Value, Count, and Refill
E	W		Write
E	WEF		Write End-of-File
M	Z		Store Zero

FOOTNOTES

1. This mnemonic is a system symbol. It must be prefixed by the character "$" whenever used.

2. This mnemonic is both an indicator mnemonic and a system symbol. It must be prefixed by the "$" whenever it is used as a system symbol in a symbolic field of some instruction. This mnemonic may also be used directly to express a Branch on Indicator instruction by being substituted for the letter "I" in any of the following four formats:

BI	Branch on Indicator
BIZ	Branch on Indicator and Zero
BZI	Branch on Zero Indicator
BZIZ	Branch on Zero Indicator and Zero

The mnemonics BI, BIZ, BZI, BZIZ are not in themselves legal operation codes. Any of the integers 0 through 63 may also be substituted for I if it is desired to designate an indicator numerically.

3. This operation code may be suffixed by the letter "I" to invoke immediate addressing.

4. This VFL operation code may have the following suffixes:

I	Immediate
N	Negative
NI	Negative Immediate

5. This operation code may be suffixed by the letter "N" to invoke the negative sign modifier.

6. This floating point operation code may be suffixed by the letter "A" to invoke the absolute sign modifier.

7. This floating point operation code may have the following suffixes:

N	Negative
A	Absolute
NA	Negative Absolute

8. Count and Branch operation may have the following suffixes:

+	Add one to value
–	Subtract one from value
H	Add half to value

9. This operation code may be used to indicate either an immediate indexing operation or the secondary operation of any VFL instruction.

10. This operation mnemonic specifies, potentially, 16 connect instructions. Four binary digits are written directly after the operation code to select a particular one of the 16 instructions. This operation code is also subject to Footnote 3.

11. This code may be used as a secondary operation with I-O select orders that are subject to end-of-operation interrupts.

12. These mnemonics are mathematical constants.

APPENDIX B

STRAP-II PSEUDO-OPERATIONS

Mnemonic	Name
BS	Backspace
CCR	Chain Counts Within Record
CD	Count Disregarding Record
CDSC	Count Disregarding Record, Skip, and Chain
CF	Count Field
CNOP	Conditional No Operation
CR	Count Within Record
CRDRUN	Card Run-Out
CW	Control Word
DD	Data Definition
DDI	Data Definition Immediate
DR	Data Reservation
DRZ	Data Reservation and Set to Zero
ECC or ODDECC	ECC (and odd parity for tape)
END	End
ERG	Erase Cap
EVEN	Even Parity No ECC (tape only)
EXT	Extract
GONG	Sound Gong
HD	High Density
KLN	Check Light On
LD	Low Density
NOECC	No ECC, Even Parity (tape only)
ODDECC	Odd Parity, ECC
ODDNEC	Odd Parity, No ECC
PRND	Print Double-spaced
PRNID	Print ID
PRNS	Print Single-spaced
PUNFUL	Punch Full Cards
PUNID	Punch ID
PUNNOR	Punch Normally
REM	Resume Error Marks
REW	Rewind
RF	Refill Field
RLF	Reserved Light Off
RLN	Reserved Light On
SCCR	Skip, Chain Counts Within Record
SCR	Skip, Count Within Record
SCD	Skip, Count Disregarding Record
SCDSC	Skip, Count Disregarding Record, Skip and Chain
SEM	Suppress Error Marks
SKIP	Skip Paper
SLC	Set Location Counter
SP	Space
SPFL	Space File
SYN	Synonym
TAIL	Tail
TILF	Tape Indicator Light Off
TLB	Terminate Loading and Branch
UNLOAD	Unload
VF	Value Field
WEF	Write End-of-File
XW	Index Word

APPENDIX C

SYMBOLIC DESCRIPTIONS AND MNEMONICS FOR IBM 7030

The following list of mnemonics may be used with Strap-1 and Strap-2 A symbolic description of the mnemonic is given to assist the programmer. The operations symbols used are defined at the start of each section. Note that the same letter ("a" and "m" for example) has a different definition for floating point and for VFL. Carefully read the definition for each set. A more detailed description of the operation is in the IBM 7030 Reference Manual. Form A22-6530.

A specific title for each mnemonic is not given in cases where the mnemonic is derived from the basic operation by changing the sign and absolute modifiers.

In the case of VFL operations, the unsigned modifier must be implied by the data referred to or be explicitly stated in a dds.

FLOATING POINT OPERATIONS

Notation for Symbolizing the Floating Point Operations OP(dds), $A_{18}(1)$

Accumulator Operands

- a = bits (0-59) of the accumulator, and the accumulator sign, bit 4 of the sign byte register.
- b = bits (60-107) of the accumulator, and the accumulator sign.
- ab = bits (0-107) of the accumulator, and the accumulator sign.
- e(a) = bits (0-11) of a.
- f(a) = bits (12-59) of a, and s(a).
- s(a) = bit 4 of the sign byte register.
- SB(a) = bits 4-7 of the sign byte register.
- Fl (a) = bits 5-7 of the sign byte register.

Storage Operands

- m = bits (0-59) of the storage word, and its sign, bit 60.
- M = L(m) = the effective address.
- e(m) = bits (0-11) of m.
- f(m) = bits (12-59) of m, and s(m).
- s(m) = bit 60 of the storage word.
- SB(m) = bits (60-63) of the storage word.
- Fl (m) = bits (61-63) of the storage word.

$FT = Factor operand; SB($FT) = bits (60-63) of $FT.
$RM = Remainder operand.

Add

+	a+m ⟶ a	1. b is unchanged.
−	a−m ⟶ a	2. Fl (a) is unchanged.
+A	a+|m| ⟶ a	
−A	a−|m| ⟶ a	

Add to Memory

M+	m+a ⟶ m	1. Fl(m) remain unchanged.
M−	m−a ⟶ m	2. The entire accumulator and SB(a) remain unchanged.
M+A	|m|+a ⟶ m	
M−A	|m|−a ⟶ m	

Add to Fraction

F+	f(ab)+f(m) ⟶ f(ab)	1. e(m) is ignored; the add is performed with e(a) on both operands.
F−	f(ab)−f(m) ⟶ f(ab)	2. The normalized mode operates in the same way as in D+.
F+A	f(ab)+|f(m)| ⟶ f(ab)	
F−A	f(ab)−|f(m)| ⟶ f(ab)	

Add to Exponent

E+	e(ab)+e(m) ⟶ |e(ab)	1. f(m) is ignored.
E−	e(ab)−e(m) ⟶ e(ab)	2. Strap−II will assemble as unnormalized unless the normalized mode is requested by referring to normalized data or by using the dds = (N).
E+A	e(ab)+|e(m)| ⟶ e(ab)	
E−A	e(ab)−|e(m)| ⟶ e(ab)	

Add Immediate to Exponent

E+I	e(ab)+e(M) ⟶ e(ab)	1. The unnormalized mode is given unless overruled by dds = (N).
E−I	e(ab)−e(M) ⟶ e(ab)	
E+AI	e(ab)+|e(M)| ⟶ e(ab)	
E−AI	e(ab)−|e(M)| ⟶ e(ab)	

Shift Fraction

SHF	f(ab)·2^{M} ⟶ f(ab)	1. Left shift if bit 11 of M = 0.
SHFN	f(ab)·2^{-M} ⟶ f(ab)	2. Right shift if bit 11 of M = 1.
SHFA	f(ab)·$2^{\|M\|}$ ⟶ f(ab)	3. The operation is not affected by the normalized modifier.
SHFNA	f(ab)·$2^{-\|M\|}$ ⟶ f(ab)	4. The exponent is not adjusted for the shift. e(a) is unchanged.
SHFL	f(ab)·2^{M} ⟶ f(ab)	5. On a right shift, zeroes are introduced in bit 12.
SHFR	f(ab)·$2^{-\|M\|}$ ⟶ f(ab)	

Double Add

D+	ab+m ⟶ ab	1. PSH indicator goes on if the exponent difference exceeds 48.
D−	ab−m ⟶ ab	
D+A	ab+|m| ⟶ ab	
D−A	ab−|m| ⟶ ab	

Add to Magnitude

+MG	R = |a|+m	1. R ⟶ a if R ≥ 0.
−MG	R = |a|−m	2. 0 ⟶ f(a) if R < 0 and e(a) is unchanged.
+MGA	R = |a|+ |m|	3. s(a) is unchanged in either case.
−MGA	R = |a|− |m|	

Double Add to Magnitude

D+MG	R = |ab|+m	1. R ⟶ ab if R ≥ 0.
D−MG	R = |ab|−m	2. 0 ⟶ f(ab) if R < O and e(a) is unchanged.
D+MGA	R = |ab|+ |m|	3. s(a) is unchanged in either case.
D−MGA	R = |ab|− |m|	

Add Magnitude to Memory

M+MG	R = m+|a|	1. R ⟶ m if s(R) = s(m)
M−MG	R = m−|a|	2. 0 ⟶ f(m) if s(R) ≠ s(m).
M+MGA	R = |m| + |a|	3. s(m) is unchanged in either case.
M−MGA	R = |m| − |a|	

Multiply

*	a·m ⟶ a	1. b in unchanged.
*N	a·−m ⟶ a	
*A	a·|m| ⟶ a	
*NA	a·−|m| ⟶ a	

Double Multiply

D*	a·m ⟶ ab	1. (108-127) of accumulator are unchanged.
D*N	a·−m ⟶ ab	
D*A	a·|m| ⟶ ab	
D*NA	a·−|m| ⟶ ab	

Multiply Factor and Add

*+	m·($FT)+ab→ab	1. The contents of $FT remain unchanged.
*N+	−m·($FT)+ab→ab	
*A+	|m|·($FT)+ab→ab	
*NA+	−|m|·($FT)+ab→ab	

Divide

/	a/m → a	1. No remainder is generated.
/N	a/−m → a	2. Quotient is 48 bits.
/A	a/|m| → a	3. Pre-normalization of the operands is independent of the normalization modifier.
/NA	a/−|m| → a	4. b is unchanged.

Reciprocal Divide

R/	m/a → a	1. Performed similarly to divide.
R/N	−m/a → a	2. b is unchanged.
R/A	|m|/a → a	
R/NA	−|m|/a → a	

Double Divide

D/	ab/m → ab	1. Remainder in $RM.
D/N	ab/−m → ab	2. 0→b except bit 60, which contains a continuation of f (a).
D/A	ab/|m| → ab	3. No rounding.
D/NA	ab/−|m| → ab	4. SB(a)→SB($RM).
		5. Result capable of being rounded in a subsequent instruction.

Store Root

SRT	$\sqrt{a}$ → m	1. ab and SB(a) are unchanged.
SNRT	$-\sqrt{a}$ → m	
SRTA	$\sqrt{\|a\|}$ → m	
SNRTA	$-\sqrt{\|a\|}$ → m	

Load

L	m → a	1. 0 → Fl(a).
LN	−m → a	2. b is unchanged.
LA	|m| → a	
LNA	−|m| → a	

Double Load

DL	m → a	1. 0 → b.
DLN	−m → a	2. 0 → Fl(a)
DLA	|m| → a	
DLNA	−|m| → a	

Load with Flag Bits

LWF	m → a	1. Fl(m) → Fl(a).
LWFN	−m → a	
LWFA	|m| → a	
LWFNA	−|m| → a	

Double Load with Flag Bits

DLWF	m → a	1. 0 → b.
DLWFN	−m → a	2. Fl(m) → Fl(a).
DLWFA	|m| → a	
DLWFNA	−|m| → a	

Load Factor

LFT	m → $FT	1. ab and SB(a) are not changed.
LFTN	−m → $FT	2. s(m) → (60)$FT.
LFTA	|m| → $FT	3. 0 → (61-63)$FT.
LFTNA	−|m| → $FT	

Store

ST	a → m	1. Fl(a) → Fl(m).
STN	−a → m	2. a is unchanged.
STA	|a| → m	
STNA	−|a| → m	

Store Rounded

SRD	a → m	1. A one is added in bit (60)b prior to the store: a and (60)b are unchanged.
SRDN	−a → m	2. Fl(a) → Fl(m).
SRDA	|a| → m	
SRDNA	−|a| → m	

Store Low Order

SLO	b → f(m)	1. e(a) - 48 → e(m).
SLON	−b → f(m)	2. Fl(a) → Fl(m).
SLOA	|b| → f(m)	3. e(a) is unchanged.
SLONA	−|b| → f(m)	

Compare

K	a:m	1. Indicators AL, AE, and AH are set as follows:
KN	a:−m	AL is set to one if a < m
KA	a:|m|	AE is set to one if a = m
KNA	a:−|m|	AH is set to one if a > m
		2. Zero exponents of different sign are considered equal.
		3. If the exponent difference is 48 the larger of the numbers is per sign and exponents regardless of fractions.

Compare for Range

KR	a:m	1. If AH is off prior to this op, no indicators will be changed.
KRN	a:−m	2. If AH is on:
KRA	a:|m|	AL is unchanged.
KRNA	a:−|m|	AE is set to one if a < m.
		AH is set to one if a ≥ m.

Compare Magnitude

KMG	a:m	1. Same as Compare, except for accumulator comparand.
KMGN	a:−m	
KMGA	a:|m|	
KMGNA	a:−|m|	

Compare Magnitude for Range

KMGR	a:m	1. Same as Compare for Range, except for accumulator comparand.
KMGRN	a:−m	
KMGRA	a:|m|	
KMGRNA	a:−|m|	

VARIABLE FIELD LENGTH OPERATIONS

Notation for Symbolizing the Variable Field Length Operations OP(dds), $A_{24}(I)$, $OF_7(I')$

Accumulator Operands

a = the accumulator operand whose:
1. Low order bit is defined by the offset;
2. Byte size is four for decimal arithmetic, eight for binary arithmetic;
3. Length includes all bits in the accumulator to the left of the offset;
4. Sign is indicated by bit four of the sign byte register.

$\bar{a}$ = the accumulator operand, a, but without sign.

a_{20} = the accumulator operand, a, with offset = 20.

Storage Operands

m = the storage operand whose:
1. High-order bit is defined by the bit address;
2. Byte size may be any number from one to eight, but is assumed to be four in the instruction lists below;
3. Length is defined by the field length in the dds;
4. Sign is bit s in the sign byte.

$\bar{m}$ = the storage operand in which all bytes are processed as data; a positive sign is assumed.

The unsigned storage operand is designated by the dds.

Bits 7.17 and 7.18 are the leftmost two bits of $LZC.

$FT = Factor Operand; s($FT) = bit 60; FL($FT) = bits (61-63).

$TR = 64-bit Transit Register.

Integer Operations

Operations which can have an immediate operand are followed by (1), except for °+.

Add

+	a+m ⟶ a	(I) 1. If the sign changes, bits to the right of the offset are complemented.
−	a−m ⟶ a	

Add To Memory

M+	m+a ⟶ m
M−	m−a ⟶ m

Add to Magnitude

+MG	R=$\bar{a}$+m	(1) 1. R⟶$\bar{a}$ if R ≥ 0.
−MG	R=$\bar{a}$−m	2. 0⟶entire accumulator if R < 0.
		3. s(a) is not changed by these operations.

Add Magnitude To Memory

M+MG	R=m+$\bar{a}$	1. R⟶m if s(R) = s(m).
M−MG	R=m−$\bar{a}$	2. 0⟶m if s(R) ≠ s(m).
		3. s(m) is not changed.

Multiply

°	a·m ⟶ a_{20}	(I) 1. Multiplication takes place only if mode = B or BU.
°N	a·−m ⟶ a_{20}	2. The decimal mode gives LTRS and 00_2 to bits 7.17 and 7.18.
		3. The length of a or m must be ≤ 48 bits in binary multiply.
		4. The portion of the accummulator not containing the product is set to zero.

Multiply Factor and Add

°+	m·($FT)+a ⟶a	(I) 1. Write: °1+ and °N1+ for an immediate operand.
°N	−m·($FT)+a⟶a	2. Multiplication takes place only if mode = B or BU.
		3. Decimal mode gives LTRS and 10_2 to bits 7.17 and 7.18.

Divide

/	a/m ⟶ a	(I) 1. Divide takes place only in the binary mode.
/N	a/−m ⟶ a	2. Decimal divide gives LTRS and 01_2 in bits 7.17 and 7.18.
		3. The remainder is placed in $RM. The remainder sign, (60) $RM, is the same as the original s(a). Fl ($RM) = 0.
		4. Bits to the right of the offset are cleared.

Load

L	m ⟶ a	(I) 1. 0⟶Fl(a).
LN	−m ⟶ a	2. The entire accumulator is cleared before the load.

Load with Flag Bits

LWF	m ⟶ a	(I) 1. Fl(m)⟶Fl(a).
LWFN	−m ⟶ a	

Load Factor

LFT	m ⟶ $FT	(1) 1. 0⟶(61 − 63) $FT.
LFTN	−m ⟶ $FT	2. The offset field is ignored.

Load Transit and Set

LTRS	m ⟶ $TR	(I) 1. Offset⟶$AOC.
LTRSN	−m ⟶ $TR	2. 11_2⟶bits 7.17 and 7.18.
		3. Indicator $BTR = 1 and $DTR = 0 if mode is B or BU. Indicator $DTR = 1 and $BTR = 0 if mode is D or DU.

Store

ST	a ⟶ m	1. SB(a)⟶SB(m).
STN	−a ⟶ m	2. If the byte size is greater than four:
		Binary: zone bits of the sign byte register are stored in SB(m).
		Decimal: zone bits of the sign byte register are stored in each byte of m.

Store Rounded

SRD	These operations are the same as the corresponding
SRDN	Store operations, except for:

a. Binary: a one is added one bit to the right of the offset, prior to the store.

b. Decimal: 0101 is added one byte to the right of the offset, prior to the store.

c. The accumulator is unchanged, even if rounding occurs.

Add One to Memory

M+1	m+1 ⟶ m	1. The one is added to the low order byte.
M−1	m−1 ⟶ m	2. The offset field is ignored.

Compare

K	a:m	(I) 1. The Compare operations set the AL, AE, and AH indicators.
KN	a:−m	AL is set to one if: a < m
		AE is set to one if: a = m
		AH is set to one if: a > m
		2. All bits to the left of the offset in the accumulator participate in the compare.

Compare for Range

KR	a:m	(1) 1. If the AH indicator is off prior to the operation, it is executed as a NOP.
KRN	a:−m	2. If AH is on:
		AL is unchanged.
		AL is set to one if a < m
		AH is set to one if a ≧ m

Compare If Equal

KE	a:m	(1) 1. If the AE indicator is off, no changes will occur.
KEN	a:−m	2. If the AE indicator is on, the indicators are set as in Compare, K.

Compare Field

KF	$\bar{a}$:m	(1) 1. The indicators are set as in Compare.
KFN	$\bar{a}$:−m	2. The length of the accumulator comparand is the same as the length of the storage comparand.
		3. The matching bits of both operands are compared.

Compare Field for Range

KFR	$\bar{a}$:m	(I)	1. The accumulator comparand is the same as in Compare Field, KF.
KFRN	$\bar{a}$:−m		2. The indicators are set as in Compare Range, KR.

Compare Field If Equal

KFE	$\bar{a}$:m	(I)	1. The accumulator comparand is the same as in Compare Field, KF.
KFEN	$\bar{a}$:−m		2. The indicators are set as in Compare If Equal, KE.

Logical Connectives OP(dds), A_{24} (I), OF_7 (I')

Note: If the operand from storage has a byte size (BS) less than eight, then eight minus BS (8 − BS) leading zeros are added to each byte from storage before the connect takes place. However, the storage operand is not changed in Cxxxx or CTxxxx.

Connect to Accumulator

$Cx_1x_2x_3x_4$ Result⟶a

Connect to Memory

$CMx_1x_2x_3x_4$ Result⟶m

Connect for Test

$CTx_1x_2x_3x_4$ Result is not stored.

$x_1x_2x_3x_4$ is a four-bit binary configuration to describe the type of connective; it is summarized:

Let: m = a bit from storage (may be an inserted leading zero if the byte size is less than 8.)
a = a bit from the accumulator corresponding to m. The accumulator byte size always = 8.
x_1 = desired result if m = 0 and a = 0
x_2 = desired result if m = 0 and a = 1
x_3 = desired result if m = 1 and a = 0
x_4 = desired result if m = 1 and a = 1

Example: C1010 (BU, 64, 4), 0 will complement the entire 128-bit accumulator.

Pseudo-Connectives

LF	(Load Field)	LF = C0011
SF	(Store Field)	SF = CM0101

Immediate Connects

To indicate immediate addressing, write: $CIx_1x_2x_3x_4$, $CTIx_1x_2x_3x_4$, and LFI.

$AOC = All ones count register.
$LZC = Left zeros count register.

After a connective operation the two registers, $AOC and $LZC contain the indicated counts of the result. Because the result may not occupy the entire accumulator, $AOC and $LZC may not give the total count of ones and left zeros of the accumulator. However, these counts always give the correct count in CM or SF.

Convert Instructions

Definitions:

a_D = accumulator in decimal, four-bit bytes with specified offset.
a_B = accumulator in binary with specified offset.
a_{B20} = accumulator in binary with offset = 20.
a_{B68} = accumulator in binary with offset = 68.
m_B = storage operand in binary with specified byte size and field length.
m_D = storage operand in decimal with specified byte size and field length.
$TR = 64-bit transit register with a sign byte in the rightmost four bits.

Note: The conversion goes: from decimal to binary if the mode given is decimal; from binary to decimal if the given mode is binary.

Convert

CV	$a_D \longrightarrow a_{B68}$	if mode = D or DU	1. In binary a field of 48 bits is used.
or	$a_{B68} \longrightarrow a_D$	if mode = B or BU	2. The entire accumulator to the left of the offset is used.
CVN	$-a_D \longrightarrow a_{B68}$		
or	$-a_{B68} \longrightarrow a_D$		

Double Convert

DCV	$a_D \longrightarrow a_{B20}$	1. In binary, a field of 96 bits is used.
or	$a_{B20} \longrightarrow a_D$	2. The entire accumulator to the left of the offset is used.
DCVN	$-a_D \longrightarrow a_{B20}$	
or	$-a_{B20} \longrightarrow a_D$	

Load Converted

LCV	$m_D \longrightarrow a_B$	(I)	1. s(m)⟶s(a)
or	$m_B \longrightarrow a_D$		2. 0⟶Fl(a)
LCVN	$-m_D \longrightarrow a_B$	(I)	3. The entire accumulator is cleared before the load.
or	$-m_B \longrightarrow a_D$		

Load Transit Converted

LTRCV	$m_D \longrightarrow \$TR_B$	(I)	1. The accumulator and offset are ignored.
or	$m_B \longrightarrow \$TR_D$		2. 0⟶Fl($TR)
LTRCVN	$-m_D \longrightarrow \$TR_B$	(I)	3. s(m)⟶s($TR)
or	$-m_B \longrightarrow \$TR_D$		4. The entire $TR is cleared before the load.

Progressive Indexing

Any VFL or Connective operation (when not immediate) may have a second operation enclosed in parentheses. The second operation may be V ± I, V ± IC or V ± ICR.

Format: $OP(OP_2)(dds), A_{24}$ (J), OF_7 (I')

Notes: 1. The original value field J is the effective address of operation.
2. A_{24} is the immediate operand specified by J in V ± I, and so on, and the value field of J is incremented by ± A_{24} according to ± I. The incrementing takes place subsequent to note I.
3. J may be $XO.

INDEXING OPERATIONS

Notation for symbolizing the Indexing Operations

Index Word Operands

J = bits (0 − 63) of the index word
V = bits (0 − 24) of J.
C = bits (28 − 45) of J.
R = bits (46 − 63) of J.

Storage Word Operands

m = bits (0 − 63) of a storage word.
V(m) = bits (0 − 24) of m if the second operand is V. (sign of V is in bit 24)
V(m) = bits (0 − 17) of m if the second operand is C or R.

Immediate Operands

m = bits (0 − 18) of the effective address if the second operand is V.
m = bits (0 − 17) of the effective address if the second operand is C or R.

Notes: 1. For clarity, the titles to the indexing and the branch operations have been omitted.

2. The indicators XF, XCZ, XVLZ, XVZ, and XVGZ are set by all of the direct and immediate index operations except KV, KC, KVI, KVNI, and KCI. These indicators are set before the refill (if any) takes place.

KV, KC,. . . .,KCI set the index compare indicators XL, XE, and XH.

Direct Index Arithmetic OP, J, A_{19} (I)

LX	m ⟶ J	A_{18}
LV	V(m) ⟶ V	1. M = A_{19} (I)
LC	V(m) ⟶ C	2. m = (M)
LR	V(m) ⟶ R	3. C_2 = The count field of J after modification
SX	J ⟶ m	1. A_{18}
SV	V ⟶ V(m)	
SC	C ⟶ V(m)	1. 0 ⟶ (18 - 24) of m.
SR	R ⟶ V(m)	I. 0 ⟶ (18 - 24) of m.
V+	V+V(m) ⟶ V	1. There is no V − etc.
V+C	V+V(m) ⟶ V C−1 ⟶ C_2	
V+CR	V+V(m) ⟶ V C−1 ⟶ C_2 (R) ⟶ (J) if $C_2 = 0$	
SVA	V ⟶ V(m)	I. V is truncated to 18, 19, or 24 bits, as is appropriate for the instruction containing V(m).
LVE	$(M)^n$ ⟶ V	I. (M) means contents of M $(M)^1$ " " " (M) . . . $(M)^n$ " " " $(M)^{n-1}$
KV ⟶ V:V(m) KC ⟶ C:V(m)		1. Indicators: XL, XE, XH are set by KV and KC. This setting is the only output of KV and KC.
RNX	J⟶(R($XO)) M ⟶ R($XO) m ⟶ J	I. Used for saving and restoring index registers.
LVS	(special format):	LVS, J, A^1, A^2, . . . , A^n
$\sum_{i=I}^{n}$	$V(A^1) \longrightarrow V(J)$	I. The sum may include any subset of the index words, each one appearing no more than once. 2. No indexing of the address field is allowed.

Immediate Index Arithmetic OP, J, A_{19}

Notes:
1. None of the immediate index instructions allow for indexing of the address. A_{19} is the effective address and is represented by A below.
2. The output of KVI, KVNI, and KCI is the setting of indicators XL, XE, and XH.

LVNI	−A ⟶ V	1. (19 - 23) of V are set to 0.
LVI	A ⟶ V	I. (19 - 24) of V are set to 0.
LCI	A ⟶ C	
LRI	A ⟶ R	
V+I	V+A ⟶ V	I.
V−I	V−A ⟶ V	I.
V+IC	V+A ⟶ V C−I ⟶ C	I.
V−IC	V−A ⟶ V C−I ⟶ C	1. A is appended by 5 zero bits for the operation.
V+ICR	V+A ⟶ V C−I ⟶ C_2 (R) ⟶ (J) if $C_2 = 0$	I.
V−ICR	V−A ⟶ V C−I ⟶ C_2 (R) ⟶ (J) if $C_2 = 0$	1.
C+I	C+A ⟶ C_2	
C−I	C−A ⟶ C_2	
KVI	(0 - 18) of V:A	I. (19 - 24) of V are compared with zeros.
KVNI	(0 - 18) of V:A	I. (19 - 23) of V are compared with zeros and (24) of V is compared with 1 (minus).
KCI	C:A	

Count and Branch Operations OP, J, B_{19} (K)

CB	C_1 - I ⟶ C_2 IC_1 + 0.32 ⟶ IC if $C_2 = 0$ M ⟶ IC if $C_2 \neq 0$	1. K may be only 0 or I. 2. M = the effective address of B_{19} (K). 3. IC_1 is the value of the instruction counter where the CB instruction is located. 4. C_1 and C_2 are the count field of J before and after the count portion of the instruction, respectively.
CBR	C_1 − I ⟶ C_2 IC_1 + 0.32 ⟶ IC and (R)⟶(J) if $C_2 = 0$ M ⟶ IC if $C_2 \neq 0$	
CBZ	C_1 − I ⟶ C_2 IC_1 + 0.32 ⟶ IC if $C_2 \neq 0$ M ⟶ IC if $C_2 = 0$	
CBRZ or CBZR	C_1 - I ⟶ C_2 IC_1 + 0.32 ⟶ IC if $C_2 \neq 0$ M ⟶ IC and (R)⟶(J) if $C_2 = 0$	

Note: In addition to the stated functions, the value field of J may be modified by placing +, −, or H after the above mnemonics. The modification of V takes place regardless of C_2 and before the refill (if any).

Example: In addition to the given functions of CB, we have:

CB	leave V alone
CB+	V + 1.0 ⟶ V
CB−	V − 1.0 ⟶ V
CBH	V + 0.32 ⟶ V

Unconditional Branch Operations: OP, A_{19} (I)

B BR	M ⟶ IC M+IC_1 + 0.32 ⟶ IC	I. The unconditional branch instructions are the only branch instructions which allow a 4 bit index field, I. The conditional branch instructions may have only a I-bit index field, K. 2. IC_1 is the value of the instruction is located (i.e., the leftmost bit of the instruction).
BE	Enable M ⟶ IC	
BD	Disable M ⟶ IC	
BEW	Enable M Wait ⟶ IC	
NOP	IC_1 + 0.32 ⟶ IC	

Branch on Bit Operations: OP, A_{24} (I), B_{19} (K)

BB	IC_1 + 0.32 ⟶ IC if $m_1 = 0$ M_2 ⟶ IC if $m_1 = 1$	1. m_1 =(A_{24}(I)), the bit being tested. 2. M_2 =B_{19}(K), the branch address. 3. K = 0 or 1; I = 1 −I5.
BZB	IC_1 + 1.0 ⟶ IC if $m_1 = 1$ M_2 ⟶ IC if $m_1 = 0$	

Note: The BB and BZB may have a suffix, Z, I, or N, which, respectively, will set m_1 to zero or to one, or negate it. This function is independent of the success of the branch. For example, the following branch on bit instructions are permissible and perform the stated functions as well as:

BB	BZB	leave m_1 alone
BBZ	BZBZ	0 ⟶ m_1
BBI	BZBI	I ⟶ m_1
BBN	BZBN	$-m_1$ ⟶ m_1

Branch on Indicator Operations BIND, B_{19} (K)

BIND	IC_1 + 0.32 ⟶ IC if ind. = 0 M ⟶ IC if ind. = 1	I. The indicators may not be set to 1 or negated with a BIND operation.
BZIND	IC_1 + 0.32 ⟶ IC if ind. = I M ⟶ IC if ind. = 0	

Notes: 1. The letters "IND" in BIND are replaced by the appropriate indicator mnemonics as shown in note 2 below.

2. The above operations can have a suffix, Z, which will cause the indicator being tested to be set to zero independently of the success of the branch. For example, BZXPOZ will set indicator XPO to zero arbitrarily. We may have: BXPO; BZXPO; BXPOZ; and BZXPOZ. The following list indicates all of the indicator mnemonics which may be used in BIND, $B_{19}(K)$, and their bit addresses.

Mnemonic	Name	Bit Address
	EQUIPMENT CHECK	
MK	Machine Check	11.0
IK	Instruction Check	11.1
IJ	Instruction Reject	11.2
EK	Exchange Control Check	11.3
	ATTENTION REQUEST	
TS	Time Signal	11.4
CPUS	CPU Signal	11.5
	INPUT-OUTPUT REJECTS	
EKJ	Exchange Check Reject	11.6
UNRJ	Unit Not Ready Reject	11.7
CBJ	Channel Busy Reject	11.8
	INPUT-OUTPUT STATUS	
EPGK	Exchange Program Check	11.9
UK	Unit Check	11.10
EE	End Exception	11.11
EOP	End of Operation	11.12
CS	Channel Signal	11.13
	(not available)	11.14
	INSTRUCTION EXCEPTION	
OP	Operation Invalid	11.15
AD	Address Invalid	11.16
USA	Unended Sequence of Addresses	11.17
EXE	Execute Exception	11.18
DS	Data Store	11.19
DF	Data Fetch	11.20
IF	Instruction Fetch	11.21
	RESULT EXCEPTION	
LC	Lost Carry	11.22
PF	Partial Field	11.23
ZD	Zero Divisor	11.24
	RESULT EXCEPTION-FLOATING POINT	
IR	Imaginary Root	11.25
LS	Lost Significance	11.26
PSH	Preparatory Shift Greater than 48	11.27
XPFP	Exponent Flag Positive	11.28
XPO	Exponent Overflow	11.29
XPH	Exponent High	11.30
XPL	Exponent Range Low	11.31
XPU	Exponent Underflow	11.32
ZM	Zero Multiply	11.33
RU	Remainder Underflow	11.34
	FLAGGING	
TF	T Flag	11.35
UF	U Flag	11.36
VF	V Flag	11.37
XF	Index Flag	11.38
	TRANSIT OPERATIONS	
BTR	Binary Transit	11.39
DTR	Decimal Transit	11.40
	PROGRAMMER INDICATORS	
PGO or PG		11.41
PG1		11.42
PG2		11.43
PG3		11.44
PG4		11.45
PG5		11.46
PG6		11.47
	INDEX RESULT	
XCZ	Index Count Zero	11.48
XVLZ	Index Value Less than Zero	11.49
XVZ	Index Value Zero	11.50
XVGZ	Index Value Greater Than Zero	11.51
XL	Index Low	11.52
XE	Index Equal	11.53
XH	Index High	11.54
	ARITHMETIC RESULT	
MOP	To-Memory Operation	11.55
RLZ	Result Less than Zero	11.56
RZ	Result Zero	11.57
RGZ	Result Greater than Zero	11.58
RN	Result Negative	11.59
AL	Accumulator Low	11.60
AE	Accumulator Equal	11.61
AH	Accumulator High	11.62
	MODE	
NM	Noisy Mode	11.63

TRANSMIT OPERATIONS: OP, J, $A_{18}(I)$, $A'_{18}(I')$

Notes:
1. Full words are transmitted in all Transmit and Swap instructions.
2. In the immediate operations, J is the count of the number of full words transmitted. J must be ≤ 16. If $J = 0$, 16 words are transmitted.
3. In the others (the direct transmission) the count field of J has the number of full words to be transmitted.

Transmit Forward

T $(M_1) \longrightarrow (M_2)$
$(M_1+1) \longrightarrow (M_2+1)$
etc.

1. M_1 is the effective address of $A_{18}(I)$
2. M_2 is the effective address of $A'_{18}(I')$

Transmit Forward Immediate

TI $(M_1) \longrightarrow (M_2)$
$(M_1+1) \longrightarrow (M_2+1)$
etc.

Transmit Backward

TB $(M_1) \longrightarrow (M_2)$
$(M_1-1) \longrightarrow (M_2-1)$
etc.

1. Both blocks are referred to in a backward direction.

Transmit Backward Immediate

TBI $(M_1) \longrightarrow (M_2)$
$(M_1-1) \longrightarrow (M_2-1)$
etc.

Swap Forward

SWAP $(M_1) \longleftrightarrow (M_2)$
$(M_1+1) \longleftrightarrow (M_2+1)$
etc.

Swap Forward Immediate

SWAPI $(M_1) \longleftrightarrow (M_2)$
$(M_1+1) \longleftrightarrow (M_2+1)$
etc.

Swap Backward

SWAPB $(M_1) \longleftrightarrow (M_2)$
$(M_1-1) \longleftrightarrow (M_2-1)$
etc.

Swap Backward Immediate

SWAPBI $(M_1) \longleftrightarrow (M_2)$
$(M_1-1) \longleftrightarrow (M_2-1)$
etc.

MISCELLANEOUS OPERATIONS: OP, $A_{19}(I)$

Store Instruction Counter If

SIC	$IC_1+1.0 \longrightarrow (0-18)$ of $A_{19}(I)$ if the following half word branch instruction is executed.	I. SIC; NOP will not store the IC.

Refill

R	$(R_M) \longrightarrow (M)$	1. R_M = refill field of word M.

Refill If Count Is Zero

RCZ $(R_M) \longrightarrow (M)$ if C field of M = 0

Execute

EX	Execute $\longrightarrow$ (M)	I. The instruction located at M is executed. 2. Control then goes to the instruction following EX.

Execute Indirect and Count

EXIC	Execute $\longrightarrow$ $(M)^1$ $(M) + I \longrightarrow (M)$	1. The instruction whose address is located in M is executed.

Store Zero

Z	$0 \longrightarrow (M)$	I. Full word of zeros.

INPUT-OUTPUT INSTRUCTIONS: OP, $A_7(I)$, $A_{18}(I')$

Locate
LOC
Select Unit
SU

$A_7(I)$ represents a channel address; $A_{18}(I')$ represents:
1. The address of one of several units attached to channel $A_7(I)$; in this case LOC or SU must be given before a RD or W addressing this channel;
2. An address on the disk specified by $A_7(I)$. LOC = SU.

Read

RD — $A_7(I)$ represents a channel address; a reading operation is initiated for this channel (or for a unit attached to this channel if more than one unit is available and has been readied by a LOC instruction). $A_{18}(I')$ is the address of a control word.

Write

W — Initiates a writing operation. Analogous to RD except that the skip flag of the control word is ignored.

Release

REL — Immediately terminates any operation in progress at the unit specified in $A_7(I)$, the channel address, or in the last unit at $A_7(I)$ selected by a LOC instruction, if $A_7(I)$ consists of more than one unit.

Copy Control Word

CCW — The current control word corresponding to the addressed channel $A_7(I)$ is sent to $A_{18}(I')$.

LOC(SEOP)
RD(SEOP)
W(SEOP)
REL(SEOP)
CTL(SEOP)
SU(SEOP)
— Same as LOC, SU, RD, W, REL, CTL except the SEOP bit in control word is set to 1; thus, program interruption on completion of an operation is suppressed, provided no exception conditions, such as unit check and end exception, are encountered.

Control

CTL — Initiates performance of certain functions at the channel indicated by $A_7(I)$, or at the last unit selected by an LOC instruction. The functions are indicated:

General I/O Unit (Standard for $A_{18}(I)$)
$A_{18}(I')$ = 016_8 Reserved Light Off
017_8 Reserved Light On
116_8 Read-Write Check Light On
057_8 ECC Mode
157_8 No ECC Mode

Card Reader and Card Punch
Standard, except $A_{18}(I') = 2$ also causes a card to be offset in the stacker.

Tape Units
Standard, but in addition:
$A_{18}(I')$ = 057_8 ECC Mode, Odd Parity
157_8 No ECC Mode, Odd Parity
156_8 No ECC Mode, Even Parity
136_8 Rewind Tape
076_8 Space Block (record)
176_8 Backspace Block (record)
077_8 Space File
177_8 Backspace File
117_8 Write Tape Mark (EOF mark)
056_8 Erase Long Gap
036_8 High-Density Mode (556 bits/inch)
037_8 Low-Density Mode (200 bits/inch)
016_8 Remove End of Tape Condition
137_8 Rewind and Unload

Inquiry Station, Printer, Console
Standard, except codes 057_8 and 157_8 are missing. On Console, $A_{18}(I') = 177_8$ causes the gong to sound.

The Current (May 1, 1961) list of STRAP II Error Messages are as follows*:

Message No.	Message	Meaning
1	MAIN S1	An improper primary op has been specified.
2	MAIN S2	An improper secondary op has been specified.
3	MAIN S3	An entry mode has been specified with a non-DD pseudo-op.
4	MAIN S4	More than one secondary op has been specified.
5	MAIN S5	More than one dds bas been specified.
6	MAIN S6	This symbol is multiple defined.
7	ASSEMBLY ERROR	There is an error in tbe assembly process.
8	MAIN S8	The internal VLE table buffer has been exceeded.
9	MAIN S9	The internal MSYTE buffer has been exceeded.
10	GETCHA4	The internal BSYST buffer is now full.
11	ASSEMBLY ERROR	There is an error in the assembly process.
12	REACHED FLAG IN NAMEXW PRIOR TO END INSTRUCTION	Output has received the flag in the index word, NAMEXW, before receiving the instruction, END.
13	FLAG NOT SET IN NAMEXW AT END INSTRUCTION	Output has received the instruction, END, but the flag in index word, NAMEXW, has not been set.
14	NAME CHECK CHARACTERS DO NOT COMPARE	The name check characters do not compare.
15	THE OUTPUT FOR THIS INSTRUCTION IS UNDETERMINED	The output for this DD instruction is undetermined.
16**	XXXXXXXCODE ERR	I/O code specified is not compatible with I/O unit specified.
17	RDR NEEDS ATTN	Card hopper has been emptied without STRAP II reaching the instruction, END.
18	UNORDER	A symbol table entry has been made unordered.
19	SYMBOL TABLE INCORRECT	There is an error in the assembly process.
21	MAIN 1	The unit begins witb an improper character.
22	MAIN 2	There is more than one leading $ on this unit.
23	MAIN 3	An illegal entry mode has been specified for a DD.
24	MAIN 4	The entry mode of tbe DD bas not been closed by a right parenthesis.
25	MAIN 5	The secondary op has not been closed by a right parenthesis.
26	MAIN 6	This op should not have a data description.
27	MAIN 7	The data description has not been closed by a right parenthesis.
28	MAIN 8	The field length of the data description is greater than 64.
29	MAIN 9	The byte size of the data description is greater than 8.
30	MAIN 10	A bit style number has been specified in the data description.
31	MAIN 11	The negative field length or byte size has been complemented.
32	MAIN 12	There are too many fields in this unit.
33	MAIN 13	This unit should not have a name.
34	MAIN 14	This instruction has been assigned a data description.
35	MAIN 15	This SYN does not have a name.
36	MAIN 16	An address field has not been specified with the SYN.
37	MAIN 17	The value of a DD was unattainable.
38	MAIN 18	A data description has not been specified with the DR.
39	MAIN 19	A character in the D field of a DD is illegal under the radix specified.
40	MAIN 20	More than one point has been used in the D field of the numeric DD.
41	EM21	Too many characters have been specified for a symbolic address.
42	MAIN 22	More than one E has been used in the D field of a numeric DD.
43	EM23	More than one $ has been used before the system symbol.
44	MAIN 24	The multiple dimensions have not been enclosed in parentheses.
45	MAIN 25	The exponent specified on a numeric DD is out of range.
46	MAIN 26	The dimension has not been closed by a right parenthesis.
48	EM 28	The system symbol specified is nonexistent.
49	EM 29	An illegal character has been specified in the level indication of the UNTAIL psuedo-op.
50	EM 30	The numeric level of untailing is greater than the current level of tailing.
51	EM 31	A null symbolic tail has been specified.
52	EM 32	The level indication of the TAIL pseudo-op. has been closed by a right parenthesis.
53	EM 33	An illegal character has been specified in the level indication of the TAIL pseudo-op.
54	EM 34	An illegal character has been used in the symbolic tail.
55	EM 35	A number has been used which is not less than the radix specified.
56	EM 36	An illegal character has been used in the address field of the PUNSYM.
57	MAIN 37	More than one parenthetical entry has been specified on the DD or more than one radix has been specified on the DD.
58	MAIN 38	GETFLD has detected an M field error.
59	MAIN 39	There is an inappropriate character string in the coded expression.
60	MAIN 40	A parenthetical entry is not allowed here.

* The messages will be modified at a later date to become more meaningful and descriptive of the error situation.

** INPUT CODE ERR—Input code specified is not compatible with input unit specified.

BINOUT CODE ERR—Output code specified is not compatible with output unit specified.

LIST CODE ERR—Output code specified is not compatible with output unit specified.

Message No.		Message	Meaning
61	41	TRUNCATION IN INDEX VALUE	The index address in the J or in the I field is larger tban the instruction field allows.
62	42	INDEX IN WRONG PLACE. IT IS IGNORED	An index has been specified in the wrong place.
63	43	SUBSCRIPT WRITTEN IN BIT-STYLE	A point has been used in a subscript.
64	44	CAN'T SUBSCRIPT CONSTANT. TRY INDEX.	A subscript has been specified with a constant.
65	45	SUBSCRIPT OR INDEX INCORRECT	Either an incorrect subscript or index has been specified.
66	46	CAN'T SUBSCRIPT SYMBOL WITH NO DDS	A subscript has been specified with a symbol that does not have a data description.
67	47	ONE SUBS. TOO MANY. LAST USED AS XR	An extra subscript has been specified.
68	48	TOO MANY SUBS. EXTRAS IGNORED	Too many subscripts have been specified. The last has been used as an index.
69	49	TOO FEW SUBS. OTHERS TAKEN AS ZERO	Too few subscripts have been specified. Other subscripts have been taken as zero.
70	50	DIVISION BY ZERO. DIVISOR IGNORED	A zero divisor has been specified.
71		GETCHA1	A non IBM card code character has been specified on the input.
72		GETCHA2	An illegal character is in the first column.
73		GETCHA3	An illegal character is in the name field.
74		GETFLD	(.0) has been interpreted as a parenthetical integer entry.
75		MAIN 41	The symbol is too long to accept the specified tail.
76		MQDALF1	The byte size should equal 12 on this DD.
77		MQDALF2	Tbe byte size should equal 8 on this DD.
78		VALUE I	A combination of bit and integer values have been specified where only an integer value is allowed.
79		MIOD I	There is an illegal sequence of MCP instructions.
80		MIOD 2	The instruction should have a name.
81		MIOD 3	The address of the IOD table of exits is null.
82		MIOD 4	This MCP instruction should not have a name.
83		EXT I	A parenthetical integer entry has been specified on a parameter of tbe EXT pseudo-op.
84		EXT 2	A parameter of the EXT pseudo-op is not followed by the correct partition character.
85		GP ERR	A parenthetical integer entry is not allowed.
86		MAIN 81	A parenthetical integer entry has been specified on the statement of a DD.
87		REMSEM1	A bit style number has been used to reference error.
88		REMSEM2	Value of error message to be suppressed or restored is not known.
89		ZERO DD	A zero base has been specified in the DD.
90		HIEX	Exponent in the DD is greater than $2^{18}-1$.
91		MQDALPX	The alphabetic DD was not terminated by the specified terminating character.
97		DEC 59	A field length greater than 64 has been specified.
98		DEC 60	A byte size greater tban 8 has been specified.
99		DEC 61	A non-allowed bit style number has been specified.
100		DEC 62	A negative field bas been complemented.
101		DEC 63	Tbe mode specified is inconsistent with the op.
102		DEC 64	No mode has been specified.
103		DEC 65	There are too many fields.
104		DEC 66	There is an error in the parenthetical integer entry.
105		DEC 67	The negative parenthetical integer entry specified has been complemented.
107		DEC 69	A non-allowed bit style number has been specified.
108		DEC 70	Negative parameters have been specified with the extract pseudo-op.
109		DEC 71	A parameter > 64 has been specified with the extract pseudo-op.
110		PASS2 1	An address < 41.0 has been specified.
111		PASS2 2	There is an error in the dd of a SYN or of a DDI.
112		NEGATIVE FIELD HAS BEEN COMPLEMENTED	The negative field specified has been complemented.
113		INDEX FIELD NOT ALLOWED	An index field is not allowed on this instruction.
114		ADDRESS FIELD HAS BEEN TRUNCATED	The address specified contains too many bits to be assembled in this instruction.
115		ONLY K FIELD ALLOWED	Only a K field is allowed.
116		ADDRESS INCLUDES BITS NOT NORMAL IN OP	The address field or fields of this instruction contains bits which shall be ignored in the actual execution of the instruction.
117		SLC CONTAINS AN INTEGER	A point has not been used in the address field of the SLC pseudo-op.
118		BIT STYLE ADDRESS NOT ALLOWED	Bit style address is not allowed.
119		BIT ADDRESSED TWICE IN LVS OR INDMK	A bit has been addressed more than once in the address field of the LVS or the INDMK instruction.
120		INSTRUCTION NOT ALLOWED IN EXT	This is an illegal statement for the EXT pseudo-op.
121		MORE THAN 1 LOC. CTR. DEP. SYMBOL	Address field contains more than one location counter symbol which may cause trouble in relocation.
122		SYMBOL ON PUNSYM NOT IN PROGRAM	A symbol specified in the address field of the PUNSYM pseudo-op is not in the program.
123		SIMAD	A field length larger than 24 has been specified on a VFL immediate op.
124		MISMUL	This is a multi-defined symbol with no contradictions.
125		CANNOT EVALUATE DDI	There is a too complicated data description for the evaluation of a DDI.
126		INCONSISTENCY IN EXT PARAMETERS	$\lvert LP-L \rvert + I \neq N$
127		CNTRCHK	There is an error in the assembly process.

IBM Technical Newsletter

System 7030

Re: Form No. C28-6129

This Newsletter No. N28-1081

Date April 15, 1963

Previous Newsletter Nos. None

ADDITIONS AND CORRECTIONS TO STRAP II REFERENCE MANUAL

In order to inform STRAP II users of all additions and corrections since the release of the STRAP manual, this bulletin covers the following:

1. System Requirements
2. New Pseudo-Operations
3. Relocatable Output
    a. Special Relocatable Pseudo-Operations
    b. Relocation Bits
    c. Relocatable Card Formats
    d. Coding Example
4. Other General Changes
5. Coding Suggestions
6. Revised Appendix B - Pseudo-Operations List
7. Additions to Appendix C - Instruction Mnemonics
8. Revised Appendix D - Error Message List
9. Revised Appendix E - Output Listing
10. Errata

SYSTEM REQUIREMENTS

Since STRAP II currently functions as a problem program under MCP control, the system requirements have been changed; the combined MCP-STRAP system requires core storage of at least 32K, a disk, console, and MCP system input and output.

NEW PSEUDO-OPERATIONS

Pseudo-operations not described in the STRAP reference manual.

1. DUPLI - Duplicate cards

    DUPLI, X, Y

The DUPLI pseudo-op will cause STRAP to repeat the next X cards Y times. Note that X refers to card images, not individual instructions; where several instructions appear on the same card, they are all duplicated. If a name appears on any card to be duplicated, it will not be included in the duplicated cards; however, a comment character in the name field will be included. X and Y must both be absolute numbers.

2. REPEAT - Duplicate cards

    REPEAT is an alternate mnemonic for DUPLI.

3. PRNTALL - Print all symbols

PRNTALL

If this pseudo-operation is specified anywhere in the program, a list of all symbols will be printed at the end of the program, with the address at which they were defined.

4. NOSEQ - No sequence numbers in binary output

NOSEQ

This pseudo-operation will cause immediate punching of any data remaining in the punch buffer, and eliminate punching the sequence number in all binary cards produced thereafter until the end of the program or a RESEQ.

5. RESEQ - Renumber sequence numbers in binary output

RESEQ

This pseudo-operation will cause immediate punching of any data remaining in the punch buffer, and begin punching sequence numbers starting with 1 in all subsequent binary cards produced thereafter until the end of the program or a NOSEQ.

The next two pseudo-ops are not recent additions, but were not included in the STRAP manual.

6. INDMK - Create one word of binary output

INDMK, A, B, C, D,.....

This pseudo-operation provides a convenient way of producing one full word of binary output beginning at a full word address, with a bit pattern as specified in the address field, bits 0-63. These integers may also be specified symbolically if desired. A sample usage is the creation of an indicator mask, using the mnemonics for the desired bits in the address field of the INDMK, e.g., INDMK, $ZM, $EXE, $IF.

7. PRNNOR - Print Normally

PRNNOR

This pseudo-operation restores printing in double-spaced format after a NOPRNT or PRNS.

RELOCATABLE OUTPUT

Special Relocatable Pseudo-Operations

1. PUNREL - Punch relocatable binary output

PUNREL

This pseudo-operation puts STRAP in relocatable mode, and must be specified before any other relocatable pseudo-operations; they will be ignored by STRAP unless it has already received a PUNREL. An assembly can be specified to produce partially relocatable and partially absolute output, since as in the case of all other punch modes, STRAP produces output in accordance with the current punch mode request.

2. ORIGIN - Punch origin card

ORIGIN, N

This command produces a special origin card to be used in execution by the loader. N may be either absolute or symbolic.

3. PUNCDC - Punch common definition card

PUNCDC

This command produces one or more special common definition cards for the loader, containing common names and sizes derived from the COMBLOCK statements immediately following PUNCDC.

4. COMBLOCK - Common block definition

A COMBLOCK, N

The name A may not exceed 8 characters. The address N refers to the size of the common block desired, and may be either absolute or symbolic. If there are more than 9 COMBLOCK statements, one or more additional cards will be punched in the same format.

5. PUNFPC - Punch FORTRAN program card

PUNFPC, S, C

This command produces a FORTRAN program card for the loader. The field S defines the program size, and C the blank common size. Either field may be absolute or symbolic. If either is not a full word, STRAP will round it up to the next higher full word. The card produced will also contain program entry points and addresses derived from the ENTER statements immediately following PUNFPC.

6. ENTER - Define entry point

A ENTER, B

The ENTER statements provide information about the program entry points to be incorporated into the FORTRAN program card. The name A, if used, may not exceed 8 characters. If it is left blank, the corresponding entry point in the FORTRAN program card will contain 8 A8 blanks. B refers to an entry point within the program. If there are more than 9 ENTER statements, one or more additional program cards will be punched in the same format except that the program size and blank common size fields will be left blank.

7. SLCRCOM - Set location counter relative to common

SLCRCOM, B

This pseudo-operation resets the location counter to zero, and includes the number of the named common B on the resulting relocatable data card, so that the BSS loader may properly position the data relative to common B. To insure that no data will be loaded into blank common, STRAP does not punch an output card where the address field is left blank following an SLCRCOM.

8. FEND - FORTRAN end card

FEND

Either END or FEND may terminate a FORTRAN program or subprogram. If FEND is used, STRAP will produce a FORTRAN branch card. If END is specified, no branch card will be punched.

Relocation Bits

A set of relocation bits is built up by STRAP describing the relocation characteristics of each half word on a binary instruction card. These relocation bits are punched consecutively following the final half word of data on the card, as determined by the bit count.

The relocation bit scheme is defined as follows:

0	No relocation
1 0	Relocation
1 0 0 ..	First 18 bits (address)
1 0 1 ..	Last 18 bits (refill)
1 0 .. 0	As lower address
1 0 .. 1 ..	As upper address
1 0 .. 1 0	Blank common
1 0 .. 1 1 i	Named common

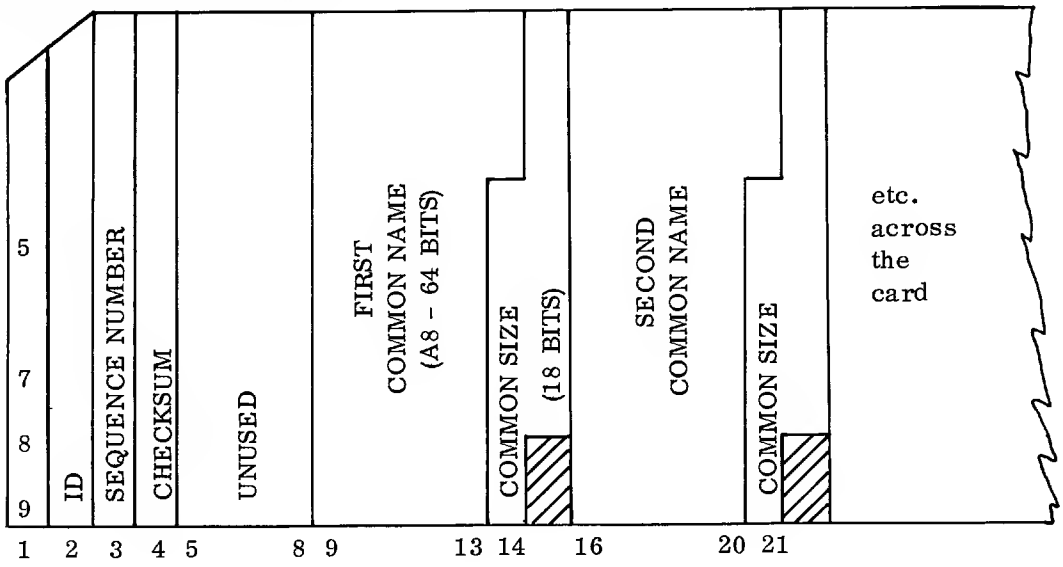
i is the number of the named common, the length of which is determined by the number of named commons. The second bit is not presently being used.

Relocatable Card Formats

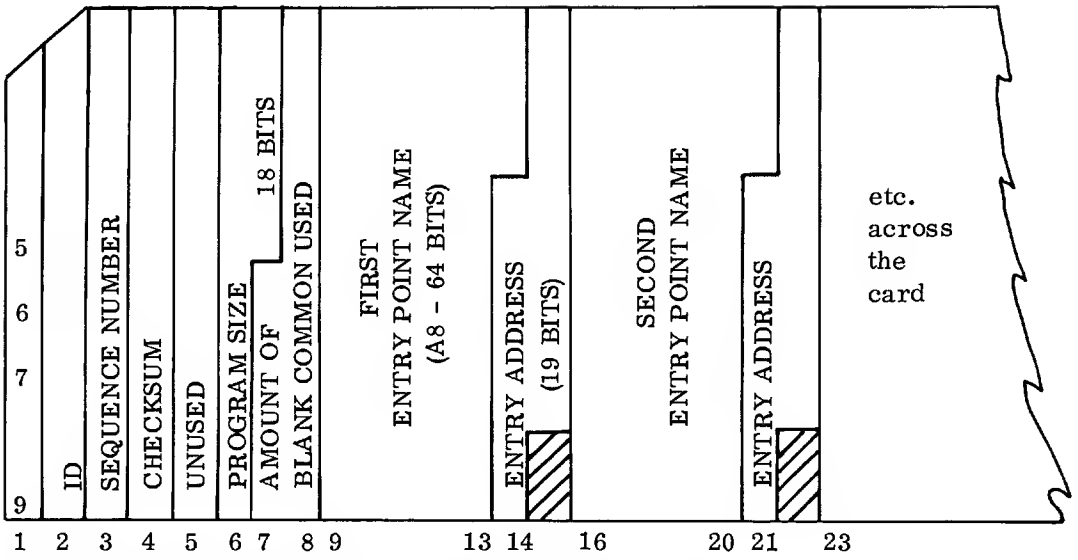
ORIGIN CARD:

Column	1 :	Hollerith O (11, 6 punches)
	2 - 9:	Octal address XXXXXX.X
	10 - 72:	Unused

COMMON DEFINITION CARD:

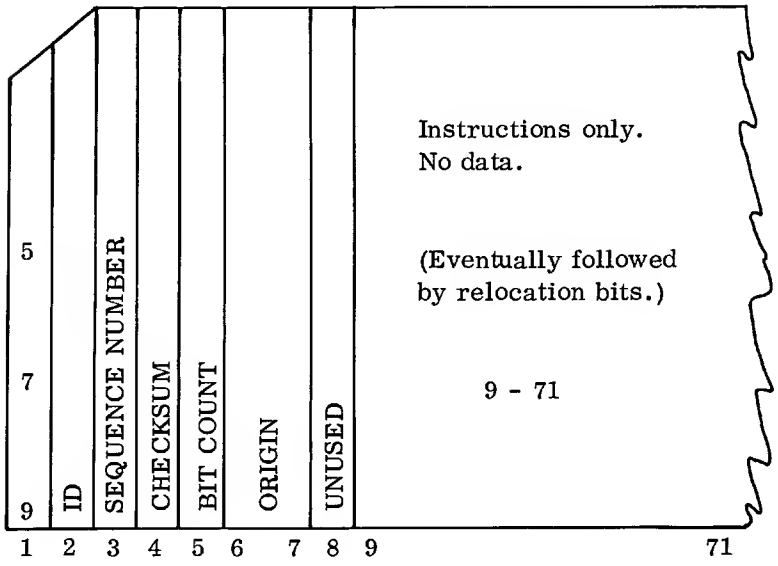


FORTRAN PROGRAM CARD:

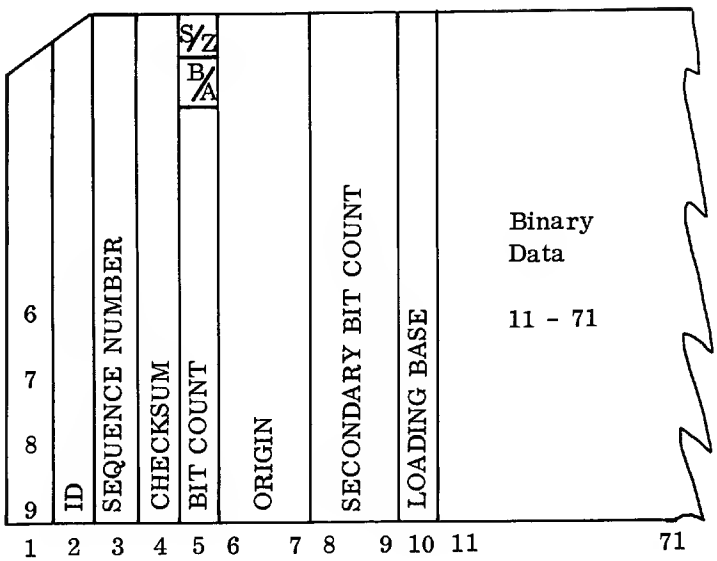


Columns 6-8 are blank on any continuation program cards.

RELOCATABLE BINARY INSTRUCTION CARD:

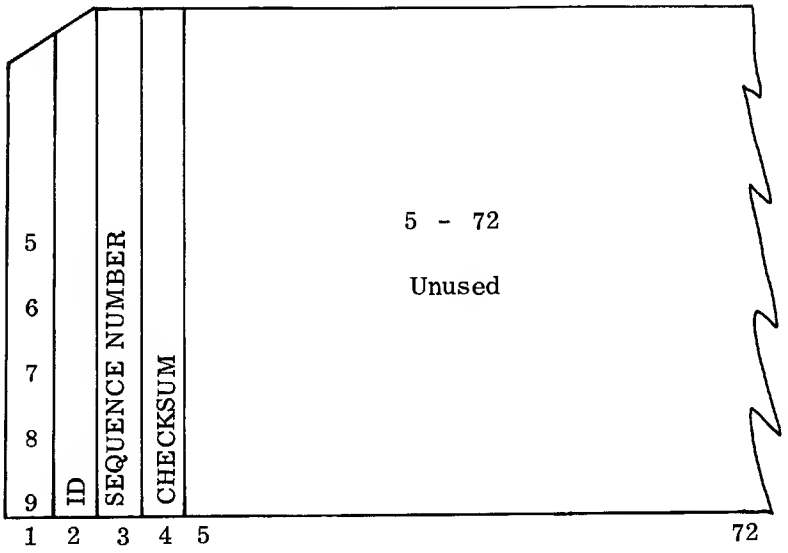


RELOCATABLE BINARY DATA CARD:



No relocation bits on this card.
Loading Base (column 10)
0-Program Data
1-1st Named Common
2-2nd Named Common
etc.
Secondary Bit Count (columns 9-10)
Bits to be zeroed/skipped before/after loading as determined from 5.0, 5.1
5.0 0-skip 1-zero
5.1 0-before 1-after

FORTRAN BRANCH CARD:



All of these card formats include the ID field of 73-80 if a PUNID has been specified.

Coding Example

```
                PRNID, AN EXAMPLE @
                PUNID, EXAMPLE @
                PUNREL @                        PUT IN RELOCATABLE MODE
                PUNFPC, FINIS, COMLAST @        PUNCH FORTRAN PROGRAM CARD
*MAIN*          ENTER, BEGIN @                  MAIN ENTRY POINT
                PUNCDC @                        PUNCH COMMON DEFINITION CARD
COM1            COMBLOCK, 100 @                 FIRST NAMED COMMON
COM2            COMBLOCK, 50 @                  SECOND NAMED COMMON
                XW, , 1 @                       LENGTH OF TRANSFER VECTOR
JOE             (AX)DD(BU), PROGRAM2X @         T.V. TO PROGRAM2
BEGIN           LINK; B, JOE @
                B, $MCP @                       FIRST EXECUTABLE INSTRUCTION
                , $EOJ @
FINIS           DR(N), 0 @                      END OF PROGRAM
                SLCRCOM @                       RELATIVE TO BLANK COMMON
A               DR(N), 50 @
COMLAST         DR(N), 0 @                      END OF BLANK COMMON
                SLCRCOM, COM1 @                 RELATIVE TO NAMED COMMON1
B               DD(N), $PI @                    DATA TO BE PLACED IN THE COMMON BLOCK
C               DR(N), 50 @
                SLCRCOM, COM2 @                 RELATIVE TO NAMED COMMON2
D               DRZ(N), 10 @
                FEND @
```

OTHER GENERAL CHANGES

1. The current version of STRAP II requires that the END (or FEND) statement be punched in columns 10-12 (10-13 for FEND).

2. When assembling a FORTRAN program, STRAP produces a type card as follows:

   B TYPE, GO, FORTRAN

3. On discovery of any program error during assembly, STRAP turns on for examination and disposition by any subsequent processor one of three bits in the communication record as follows:

   7.41_{10}: For undefined or multiply defined symbols or contagious errors.
   7.42_{10}: For all serious error messages (see revised Appendix D).
   7.43_{10}: For all other error messages.

   In addition, since it would be of questionable value to execute the GO phase of a COMPILGO assembly containing serious errors, STRAP sets on the REJECT bit in the communication record whenever it discovers either of the first two types of errors in a COMPILGO assembly with STRAP as the last member of the chain. An error message is printed on the system output, and MCP will reject the GO portion of the job.

4. With reference to multiply defined symbols, STRAP now prints the MULTI error flag on the output listing line immediately preceding both definitions. In addition, the CONTAG flag is printed immediately preceding an instruction which references a multiply defined symbol.

5. The MCP pseudo-operations are included in the STRAP system symbol table, and may be assembled as with any other system symbol, by preceding the reference with a $; e.g., $WAIT, $ABEOJ.

6. The output listing format has been somewhat changed, and a revised version of page 19 in the STRAP manual is included in Appendix E.

CODING SUGGESTIONS

Three simple coding hints are suggested, which will increase assembly speed.

1. Pack as many instructions as will fit on a card.

2. If there is only one instruction on a card or blank columns following the last instruction, put a comment mark (4-8 punch) immediately after the last character of the final instruction.

3. Use an absolute number instead of a system symbol in a J field only.

```
LX, 5, BOX @ is faster than
LX, $5, BOX @ or
LX, $X5, BOX @
```

REVISED APPENDIX B - Pseudo-Operation List

The following information should replace Appendix B:

Additional Instructions and Pseudo-Ops Accepted as STRAP II PRIMARY OPERATIONS

Pseudo-Ops

COMBLOCK	Common Block Definition
CNOP	Conditional No Operation
DDI	Data Definition Immediate
DR	Data Reservation
DRZ	Data Reservation and Set to Zero
DUPLI	Duplicate Input
END	End
ENTER	Define Entry Point
EXT	Extract
FEND	FORTRAN End
LINK	Link
NOPRNT	No Printing of Listing
NOPUN	Binary Output Suppressed, Both Cards and Disk
NOSEQ	No Sequence Number to be Punched in Binary Card(s)
ORIGIN	Origin
PRND	Print Double-Spaced
PRNID	Print ID
PRNNOR	Print Normally
PRNS	Print Single-Spaced
PRNTALL	Print All Symbol(s) Used in Program
PUNALL	Punch SYN Card(s) for All Symbol(s)
PUNCDC	Punch Common Definition Card
PUNFPC	Punch FORTRAN Program Card
PUNFUL	Punch Full Binary Card(s)
PUNID	Punch ID in Binary Card(s)
PUNNOR	Punch Normally
PUNORG	Punch Origin Binary Card(s)
PUNREL	Punch Relocatable Binary Card(s)
PUNSYM	Punch Syn Symbolic Card(s)
REM	Restore Error Message
REPEAT	Duplicate Input
RESEQ	Restore Punching Sequence Number in Binary Card(s)
SEM	Suppress Error Message
SKIP	Skip paper
SLC	Set Location Counter
SLCR	Set Location Counter Relative
SLCRCOM	Set Location Counter Relative to Common
SPNUS	Suppress Printing Not Used Symbol List
SYN	Synonym

Pseudo-Ops

TAIL	Tail
TLB	Terminate Loading and Branch
UNTAIL	Untail

General Instructions		MCP Instructions
CF	Count Field	IOD
CW	Control Word	REEL
DD	Data Definition	
INDMK	Indicator Mask	
RF	Refill Field	
VF	Value Field	
XW	Index Word	

Input-Output Instructions: OP, $A_7(I)$ where $A_7(I)$ represents a channel address and the unit affected is the last unit selected by a LOC instruction.

BS	Backspace
BSSEOP	Backspace, Suppress End of Operation Interrupt
BSFL	Backspace File
BSFLSEOP	
ECC	ECC (and odd parity for tape)
ECCSEOP	
ERG	Erase Gap
ERGSEOP	
EVEN	Even Parity No ECC (tape only)
EVENSEOP	
GONG	Sound Gong
GONGSEOP	
HD	High Density
HDSEOP	
KLN	Check Light On
LD	Low Density
LDSEOP	
NOECC	No ECC, EVEN Parity (tape only)
ODD	Odd Parity, No ECC
ODDSEOP	
ODDECC	Odd Parity, ECC
ODDNEC	Odd Parity, No ECC
RLF	Reserved Light Off
RLFSEOP	
RLN	Reserved Light On
RLNSEOP	
RWDUNL	Rewind and Unload

SP	Space
SPSEOP	
SPFL	Space File
SPFLSEOP	
TILF	Tape Indicator Light Off
UNLOAD	Unload
WEF	Write End-of-File
WEFSEOP	

SECONDARY OPERATIONS

CCR	Chain Counts within Record
CD	Count Disregarding Record
CDSC	Count Disregarding Record, Skip, and Chain
CR	Count within Record
SCCR	Skip, Chain Counts within Record
SCR	Skip
SCD	Skip, Count Disregarding Record
SCDSC	Skip, Count Disregarding Record, Skip, and Chain

ADDITIONS TO APPENDIX C - Instruction Mnemonics

The following additional mnemonics should be included in Appendix C:

	Under these headings	include
Floating Point	Add	+N, -N, +NA, -NA, -A
	Add to Memory	M+N, M-N, M+NA, M-NA
	Add to Fraction	F+N, F-N, F+NA, F-NA
	Add to Exponent	E+N, E-N, E+NA, E-NA
	Double Add	D+N, D-N, D+NA, D-NA
	Double Add to Magnitude	D+NMG, D-NMG
	Add Magnitude to Memory	M+NMG, M-NMG
Variable Field Length	Add	+I
	Add to Magnitude	+NMG, -NMG, +MGA, +MGI
	Multiply	*I, *NI
	Divide	/I, /NI
	Load	LI, LNI
	Load with Flag Bits	LNFI, LWFNI
	Load Factor	LFTI, LFTNI
	Load Transit and Set	LTRSI, LTRSNI
	Add One to Memory	M+N1, M-N1
	Compare	KI, KNI
	Compare for Range	KRI, KRNI
	Compare If Equal	KEI, KENI
	Compare Field	KFI, KFNI
	Compare Field for Range	KFRI, KFRNI
	Compare Field If Equal	KFEI, KFENI
	Connect to Accumulator	$CIX_1X_2X_3X_4$
	Connect for Test	$CTIX_1X_2X_3X_4$
	Pseudo-Connectives	LFI
	Load Converted	LCVI, LCVNI
	Load Transit Converted	LTRCVI, LTRCVNI
Input-Output	Card Runout	CRDRUN (SEOP)
	Rewind	REW (SEOP)
New Instruction	Store Multiply Register	STM STMN STMNA STMA
	Load Multiply Register	LMR LMRN LMRNA LMRNI LMRA LMRI

REVISED APPENDIX D - Error Message List

The following list of STRAP error messages should replace Appendix D. The number at the left of the message may be used if the programmer wishes to SEM any of the messages. Messages 1-18 are considered as serious error messages by STRAP, and it is strongly recommended that they should not be SEM'ed.

The error list is printed at the end of the listing, in sequence by page and line number. On the listing itself an extra asterisk is printed following the line number and asterisk on all lines containing an error.

1	ILLEGAL OPERATION CODE
2	ILLEGAL SECONDARY OP CODE
3	ENTRY MODE WITH NON-DD OPERATION
4	MORE THAN ONE SECONDARY OPERATION
5	MORE THAN ONE DDS
6	PASS 2A AND 2B LOCATION COUNTER DOES NOT AGREE
7	ASSEMBLY ERROR
8	SYMBOL TABLE EXCEEDED
9	SYMBOL BUFFER EXCEEDED
10	STATEMENT BUFFER EXCEEDED
11	SPARE
12	REACHED END OF NAME FILE BEFORE END INSTRUCTION
13	MORE NAMES IN NAME FILE AFTER END INSTRUCTION
14	ERROR ON NAME SEQUENCE
15	SPARE
16	SPARE
17	SPARE
18	SYMBOL TABLE ENTRY UNORDERED
19 - 28	SPARE MESSAGES
29	IMPROPER 1ST CHAR
30	MORE THAN ONE $
31	ILLEGAL ENTRY MODE
32	ENTRY MODE NOT CLOSED BY RIGHT PAREN
33	2NDARY OP NOT CLOSED BY RIGHT PAREN
34	THIS OP SHOULD NOT HAVE DDS
35	DDS NOT CLOSED BY RT. PAREN
36	FIELD LENGTH GREATER THAN 64
37	BYTE SIZE GREATER THAN 8
38	BIT STYLE NO. IN DDS
39	NEG. FL OR BS HAS BEEN COMPLEMENTED
40	EXTRA FIELDS
41	SHOULD HAVE NO NAME
42	STRAP ASSIGNED DDS
43	SYN WITHOUT A NAME
44	SYN WITHOUT AN ADDRESS FIELD
45	UNATTAINABLE VALUE
46	DR OR DD WITHOUT DDS
47	CHAR ILLEGAL IN RADIX SPEC.
48	MORE THAN ONE POINT
49	SYMBOL IS TOO LONG
50	MORE THAN ONE E IN NUM. DD
51	MORE THAN 1 $ IN SYSTEM SYM.
52	MULTIPLE DIMENSIONS NOT IN PAREN

53	VALUE ROUNDED TO FULL WORD
54	DIMENSION NOT CLOSED BY RIGHT PAREN
55	NO COMMA AFTER PUNID
56	NON-EXISTENT SYSTEM SYMBOL
57	PSEUDO LOC. CTR. TOO HIGH
58	UNTAIL LEVEL MORE THAN TAIL
59	NULL TAIL
60	TAIL LEVEL NOT CLOSED BY RIGHT PAREN
61	ILLEGAL TAIL LEVEL CHARACTER
62	ILLEGAL CHARACTER IN TAIL
63	DIGIT INCORRECT FOR RADIX
64	ILLEGAL CHARACTER IN PUNSYM
65	MORE THAN 1 RADIX OR PAREN. ENTRY
66	SYNTAX ERROR
67	INAPPROPRIATE CHAR.
68	GP ERROR
69	TRUNCATION IN INDEX VALUE
70	INDEX IN WRONG PLACE, IT IS IGNORED
71	SUBSCRIPT WRITTEN AS BIT
72	CANT SUBSCRIPT CONSTANT, TRY INDEX
73	SUBSCRIPT OR INDEX INCORRECT
74	CANNOT SUBSCRIPT SYMBOL WITH NO DDS
75	1 SUBS. TOO MANY, LAST USED AS INDEX
76	TOO MANY SUBSCRIPTS, EXTRAS IGNORED
77	TOO FEW SUBSCRIPTS, OTHER TAKEN 0
78	DIVISION BY ZERO, DIVISOR IGNORED
79	INCORRECT CARD CODE CHAR.
80	ILLEGAL CHAR. IN FIRST COL.
81	ILLEGAL CHAR. IN NAME FIELD
82	(.0) HAS BEEN INTERPRETED AS PAREN INTEGER
83	SYMBOL TOO LONG FOR SPECIFIED TAIL
84	CC ENTRY MODE WITHOUT BS 12
85	BS NOT 8
86	ONLY INTEGER VALUES ALLOWED
87	THE FL IS GREATER THAN 64
88	BYTE SIZE GREATER THAN 8
89	BIT TYPE NOT ALLOW.
90	NEG. FIELD HAS BEEN COMPLEMENTED
91	MODE INCONSISTENT WITH OP
92	NO MODE
93	TOO MANY FIELDS
94	ERROR IN GP
95	NEGATIVE GP HAS BEEN COMPLEMENTED
96	NO FIELDS, STATEMENT IGNORED
97	BIT TYPE UNUSUAL
98	NEGATIVE PARAMETERS ON EXT
99	PARAMETER GREATER THAN 64
100	ADDR LESS THAN 33.
101	ERR IN DDS
102	ILLEGAL SEQ. OF MCP CARDS
103	IOD CARD SHOULD HAVE NAME
104	I/O TBL OF EXITS ADDR NULL
105	REEL CARD DOESNT NEED NAME
106	MISSING FIELD
107	EXT NOT FOLLOWED BY CORRECT PARTITION CHAR
108	GP IS NOT ALLOWED

109	RADIX SPECIFIED AT THE END OF A DD
110	BIT STYLE NUMBER USED TO REFER. ERR
111	ERR MESSAGE SPECIFIED IS UNKNOWN
112	0 BASE IN DD
113	EXP. NOT IN RANGE
114	NEGATIVE FIELD HAS BEEN COMP.
115	INDEX NOT ALLOWED
116	ADDRESS FIELD HAS BEEN TRUNCATED
117	ONLY K FIELD ALLOW.
118	ADDR INCLUDES BITS NOT NORMAL IN OP
119	SLC HAS AN INTEGER
120	BIT TYPE ADDR UNUSUAL HERE
121	BIT ADDRESSED TWICE IN LVS OR INDMK
122	PSEUDO-OP SPEC. IN EXT--ZEROES SUPPLIED
123	MORE THAN 1 LOC. CTR. DEP. SYMBOL
124	SYM ON PUNSYM NOT IN PROG.
125	NO TERMINATING CHARACTER
126	FORCED COMMENT CARD
127	PUNID IN PUNFUL SEQUENCE--STATE. IGNORED
128	MULTIPLY ASSUMED
129	OP OR SS CHANGED INTO LEGAL MNEMONICS
130	FIELD LENGTH MORE THAN 24
131	MULT. DEFINED SYM WITH NO CONTRADICTION
132	CANNOT EVALUATE DDI
133	INCONSISTENCY IN EXT PARAMETERS
134	PSEUDO LOC. COUNTER CHECK
135	BIT ADDR IN DR(Z)
136	RELOCATABLE PSEUDO-OP, NOT IN PUNREL MODE
137	COMBLOCK STATEMENT WHEN NOT IN PUNCDC MODE
138	ENTER STATEMENT, NOT IN PUNFPC MODE
139	NAME LONGER THAN 8 CH. ON COMBLOCK OR ENTER
140	NO FIELDS ON RELOCATABLE STATEMENT
141	RELOCATABLE STATEMENT NEEDS A NAME
142	COMMON NAME UNDEFINED
143	EXIT ADDR IS NULL
144	NO B ON IOD
145	FIELD IS ZERO
146	REPEAT PSEUDO-OP WITHIN REPEAT BLOCK--IGNORED
147	PARAMETER EXCEEDS MAXIMUM ALLOWED
148	REPEAT PSEUDO-OP ILLEGAL HERE
149	SPECIAL SYSTEM SYMBOL IN NON DD OP
150	MISSING COMMA

```
 TIME CLOCK C0005162   PUNID              CARD 10            DIST. OF 3/01          3/01        PAGE NUMBER    1

LINE   LOCATION            BINARY OUTPUT          NAME       STATEMENT                                LOCATION     000100

CC0100.00 LOWER MEMORY BOUND

CC0115.00 UPPER MEMORY BOUND

  1* 000100.00                                              SEC,64.    -
  2*                                                        PRNS -
  3* 017777.00+     +00000000        NULL          NEXT     SYN,(8)17777.0  -
  4* 000100.00      000112.16 10                   ABLE     LX,7,INDEX   -                    LOAD INDEX
  5* 000100.40      000113.20 80  600000.20 50              L(BU),BAKER  -
  6* 000101.40      000000.10 87  204000.20 00   CHARLIE   ST(BU,4)(V+IC),.8($7) -
  7* 000102.40      000101.70 40                           BZXCZ,CHARLIE -
  8* 000103.00      000107.00 60                           L(N),OOG -
  9* 000103.40      000110.00 20                           +(N),OOG+1. -
 10* 000104.00      000111.00 E0                           ST(N),FOX -
 11* 000104.40      000113.34 80  030000.20 50             L(BU,24),SOME VERY LONG NAME -
 12* 000105.40      000113.64 80  030000.20 00             ST(BU,24),SOME OTHER LONG NAME-    TESTING LONG COMMENTS THAT
 13*                                              -        WILL CARRY OVER TO THE NEXT LINE
 14* 000106.40      017777.10 00                           B,NEXT -
 15* 000107.00     0007+ 6777700000000000 +000   OOG      DD(N),28671X7,183007S7 -
 16* 000110.00     C022+ 5456370000000000 +TUV
 17* 000111.00 *    000001.00                     FOX      DR(BU),1-
 18* 000112.00    000113.00+ 000 000004 000000    INDEX    XW,ZEBRA,4-
 19* 000113.00 *    000000.20                     ZEBRA    OR(BU,4),4-
 20* 000113.20                          5703      BAKER    (8)DD(BU,12),5703 -
 21* 000113.34                      00067777      SOME VERY LONG NAME
 22*                                                        OD(BU,24),28671
 23* 000113.64 *    000000.30                     SOME OTHER LONG NAME
 24*                                                        DR(BU,24),1
 25* 000114.14 *    000100.00                              END,ABLE -
```

ERRATA

The introduction to the STRAP manual gives alternate system requirements for running STRAP without a disk. Since STRAP now functions as a problem program of MCP, this is no longer possible. The current system requirements are listed in this bulletin.

page 1 - column 2 - line 1

not Spool Tapes
but System Input and System Output Tapes

page 1 - column 2 - line 12

eliminate and one new restriction on the use of radix 16.

page 3 - column 2 - line 36

not BZM, ERROR($7)
but BZM, ERROR($1)

page 3 - column 2 - line 41

not by index register 7
but by index register 1

page 4 - column 1 - line 36

not BB, ONEBIT($5), FIXUP($9)
but BB, ONEBIT($5), FIXUP($1)

page 4 - column 1 - line 41

not ter 9.
but ter 1.

page 6 - column 2 - line 4

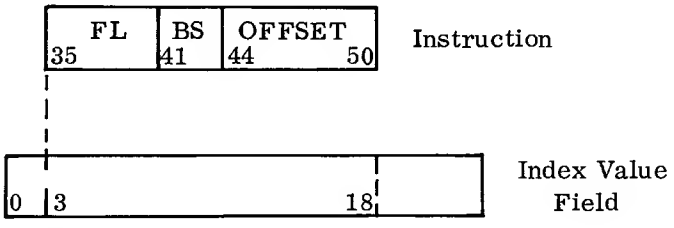
The sentence should read: The dds immediately follows the operation mnemonic, except in progressive indexing, where it may precede or follow the secondary operation.

page 10 - column 2 - line 32

not thus complied.
but thus compiled.

page 12 - column 2

The diagram showing index modification of the second half word is out of proportion. The corrected diagram is:



page 18 - column 2 - line 43

not lines 14 and 15
but lines 15 and 16

page 18 - column 2 - line 50

not line 17
but line 18

page 19	This page is out of date. A new sample output listing page is found in the section headed "General Changes".
page 20 - column 2 - line 4	not 19 or 20 but 20 or 21
page 20 - column 2 - line 14	not (see lines 11 and 12) but (see lines 12 and 13)
page 20 - column 1 - line 35	not beginning of the listing but end of the listing
page 20 - column 2 - line 49	Refer to the section headed "General Changes", Item 4 for a change in treatment of MULTI error flags.
page 21 - column 2 - line 5	not value 500.0 but value 1000.0
page 22 - column 2 - line 36	not through 72 but through 71
page 22 - column 2 - line 38	not 0 to 748 but 0 to 736
page 23 - column 2 - line 4	not data columns (5.4-7.11) but data columns (5.4-71.11)
page 23 - column 2 - line 30	The sentence beginning "If no address" is incorrect. If no address is specified on the END or TLB, $41._8$ is used.
page 23 - column 2 - line 36f.	The format indicated is incorrect. The format should read: 1.0-1.11 Code column (branch card - 1.8, 1.9, 1.11 punches) 2.0-2.11 Identification number (binary) 3.0-3.11 Sequence number (binary) 4.0-4.11 Checksum 5.0-5.11 Not presently used 6.0-7.11 24-bit transfer address
page 23 - column 1 - line 34f.	not There are only two ... and the IQS entry mode. but There are three entry modes that fall into this category, the A entry mode, the IQS entry mode, and the CC entry mode.

page 25 - column 2 - line 6

Eliminate the paragraph referring to a restriction on the use of radix 16.

page 26 - column 1 - line 27

not or data field, is it
but or data field, it is

page 28 - column 1 - line 1

not division (/).
but division (/), and certain exponentiation(**).

page 28 - column 1 - line 4

not and subtractions are completed.
but and subtractions are completed. Exponentation is allowed if the integer exponent is in the range

$$2^{-18} < \text{Integer exponent} < 2^{+18}$$

page 36 - column 1 - line 28

not M+(BU), (200.0 * 50.0).
but M+(BU), (200.0 * 50).

page 36 - column 1 - line 40

Additional pseudo-operations have been itemized in this bulletin.

page 36 - column 2 - line 9f.

The following sentence should be deleted:
At the conclusion ... in the middle of one assembly.

page 36 - column 2 - line 23

not at the beginning
but at the end

page 37 - column 1 - line 5

The sentence should be: Punching remains suppressed until another punch pseudo-operation (PUNNOR, PUNFUL, PUNORG or PUNREL) is encountered.

page 37 - column 1 - line 41

not columns 44-55
but columns 46-53

page 37 - column 2 - line 28

not As many as 256
but As many as 255

page 38 - column 2 - line 4

not A EXT, (I, J, COUNT) STATEMENT
but A EXT (I, J, COUNT) STATEMENT

page 39 - column 2 - line 40

not in statement
but in statements

page 39 - column 2 - line 43

not the numbers identifying
but the absolute numbers identifying

page 40 - column 1 - line 8	not suppressed. but suppressed. Again the number of each message must be in absolute.
page 43	Insert in its place in the list this entry: $ INF 12 ∞ (infinity)
page 46	This page is not up to date. See new Appendix B in this bulletin.
page 47f.	Appendix C is not up to date. Additional mnemonics are itemized in this bulletin.
page 47 - column 2 - line 41	not 1. b in unchanged. but 1. b is unchanged.
page 54f.	A new list of error messages is included in this bulletin.

Printed in U.S.A. C28-6129

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York